BUILDERFORMER: A PANOPTIC SEGMENTATION APPROACH TO AUTOMATIC FLOOR PLAN ANALYSIS USING VISION TRANSFORMERS

Mestiyage Don Pathum Pramuditha Goonathilake

218525N

Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

July 2024

BUILDERFORMER: A PANOPTIC SEGMENTATION APPROACH TO AUTOMATIC FLOOR PLAN ANALYSIS USING VISION TRANSFORMERS

Mestiyage Don Pathum Pramuditha Goonathilake

218525N

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree

Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

July 2024

DECLARATION

I declare that this is my own work and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date: 01/07/2024

The above candidate has carried out research for the Master's thesis/dissertation under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Dr. ALARR Thanuja

Signature of the Supervisor: Date: 01/07/2024

DEDICATION

I would like to express my heartfelt dedication of this thesis to my parents, who have served as the pioneering figures and unwavering pillars that have provided support and inspiration throughout my educational journey. Furthermore, I am deeply grateful to extend this dedication to all the lecturers of the Department of Computational Mathematics, Faculty of Information Technology, University of Moratuwa.

ACKNOWLEDGEMENT

I convey my gratitude to everyone who has contributed to completing this research project. Their constant support, guidance and encouragement have greatly benefited me.

Firstly, I want to express my heartfelt gratitude to my supervisor, Dr. ALARR Thanuja, Senior Lecturer in the Department of Computational Mathematics at the Faculty of Information Technology, University of Moratuwa. I thank him for his expert advice, exceptional mentoring, and continuous support. He was instrumental in shaping the direction and quality of this research through his deep knowledge, insightful feedback, and constant commitment to academic excellence.

In addition, I would like to extend my deepest gratitude to my family and friends for their unwavering support. Their continuous encouragement, understanding, and support are determined as a consistent source of motivation and inspiration throughout the research journey. They have been instrumental in my belief that I can do it, and their constant support at times of failure has been indispensable. I am deeply grateful for the loyalty they have shown me.

Finally, I would also like to thank BuilderBid founder Mr. Bo Dillon for accepting and developing my idea to integrate this research into their production-grade software, which will help them streamline the takeoff process of floor plans. I have been able to focus on the correct direction thanks to his immense feedback throughout the research project.

Throughout this research, I referred to numerous research publications. I sincerely appreciate the authors of these works' notable contributions to the field, which greatly enhanced the scope and quality of this project.

ABSTRACT

Automatic floor plan analysis, rooted in computer vision and pattern recognition, seeks to extract meaningful insights from architectural and interior design drawings. Its significance spans across various sectors, including architecture, interior design, real estate, and construction. However, conventional methods, whether rule-based or learning-based, encounter constraints such as manual annotation requirements and the challenge of achieving precise segmentation. This research project endeavors to evaluate the efficacy of vision transformers in tasks crucial to floor plan analysis, including identifying unique objects like doors and windows, extracting line segments to represent room layouts, and segmenting object areas such as room spaces. A novel panoptic segmentation approach is proposed to achieve a holistic understanding of scenes by integrating object detection with image segmentation. This approach involves identifying and delineating individual object instances within regions of interest and subsequently segmenting images into meaningful regions. Key to this methodology is the utilization of the Segment Anything Model for segmentation, guided by input prompts generated through object detection models. The proposed approach introduces a mask classification mechanism within the segmentation stage, diverging from conventional methods of assigning semantic labels to individual pixels. The effectiveness of this approach is demonstrated through promising results obtained on existing datasets. Real-time object detection models like YOLOv8 are employed alongside end-to-end object detection models like RT-DETR to provide visual prompts for the Segment Anything Model. This innovative approach is prepared to revolutionize automatic floor plan analysis in real-world applications by bridging the gap between object detection and image segmentation. As the final step of this research project, the proposed methodology integrates into a web application, enabling users to visualize results for various floor plan types, thus providing a practical tool for professionals in architecture, interior design, real estate, and construction to analyze floor plans with enhanced accuracy and efficiency.

Keywords: Automatic Floor Plan Analysis, Panoptic Segmentation, Real-Time Object Detection, Visual Prompt Engineering, Vision Transformers

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
API	Application Programming Interface
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
DL	Deep Learning
GAN	Generative Adversarial Network
GNN	Graph Neural Network
GPU	Graphics Processing Unit
ML	Machine Learning
RAG	Room Adjacency Graph
RT-DETR	Real-Time Detection Transformer
SAM	Segment Anything Model
SG	Shape Grammar
SOTA	State Of The Art
VIT	Vision Transformer
YOLO	You Only Look Once

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Prolegomena

Architectural floor plans serve as fundamental blueprints in the construction and real estate industries, providing detailed insight into the design and functioning of interior spaces [1]. Designers and engineers use abbreviations and symbols on floor plans to indicate the layout, scale, and external characteristics of construction sites. These notations can be categorized as Geometry, which pertains to the dimensions and shape of elements; Topology, which deals with the connection between building components; and Semantics, which encompasses extra attributes like room function [2]. Floor plans usually consist of dimension lines, furniture, grids, symbols, text, walls, and windows. Analyzing floor plans automatically poses significant challenges due to the complexities and interdependencies among these elements [1].

Floor plans are primarily used by the construction industry, which is currently facing slow or non-existent growth in major economies. To address this decline in productivity, extensive research has been conducted to optimize the design and construction process, control costs, and improve efficiency [3]. Notably, professionals in the construction industry, including builders and contractors, spend approximately two-thirds of their time estimating costs, often relying on software for floor plan analysis. However, this manual analysis process can be time-consuming and exhaustive, usually taking days [4]. Consequently, there is a clear need for an automated procedure to analyze floor plans comprehensively.

Automatically recognizing and analyzing the constituent parts of rasterized floor plan images has long been a challenge in computer vision. This task involves four primary obstacles. Firstly, architectural and engineering firms use different notations, resulting in color variations, line thickness, and symbols [5]. Secondly, rasterized floor plans often include intricate and unclear architectural drawings [6]. Thirdly, floor plans must adhere to high-level semantic, topological, and geometric constraints. Finally, examples may have varying room arrangements and floor plan layouts.

The research objective of the floor plan analysis is to extract relevant data from architectural drawings automatically. This includes identifying non-structural

components and walls, locating and classifying different rooms, and creating a digital blueprint in 2D and 3D dimensions [6]. However, traditional approaches like rule-based and learning-based methods face limitations concerning manual annotation requirements and segmentation accuracy. Manual annotation is highly time-consuming and labor-intensive, while learning-based methods struggle to achieve precise segmentation results.

All the above approaches focus solely on individual tasks and do not consider the comprehensive analysis of the floor plan. A thorough analysis should provide detailed information about each room, including its area, wall lines, and objects. Therefore, there is a pressing need for a comprehensive analysis of floor plans. To tackle these challenges, a solution must accurately segment and comprehend floor plans without relying heavily on manual annotation. The objective is to minimize the need for manual annotation, improve segmentation accuracy, and offer an automated solution with standardized evaluation metrics.

In this research project, I present BuilderFormer: A Panoptic Segmentation Approach to Automatic Floor Plan Analysis using Vision Transformers, harnessing the power of cutting-edge vision transformers in computer vision tasks. The goal is to assess the effectiveness of vision transformers in identifying unique objects like doors and windows, extracting line segments to capture room layouts, and segmenting object areas such as room spaces.

With the success of the transformer framework in natural language processing [7], there is a growing interest in exploring its suitability for computer vision applications, such as object detection and segmentation. Transformers, known for their ability to reason global relationships among tokens without predefined graph connectivity, have demonstrated potential in these vision tasks. This is particularly relevant for floor plan analysis, as vector graphics representing floor plans are well-suited for self-attention mechanisms due to their lack of handcrafted features and structure.

This chapter is organized to present the following: aim and objectives, background and motivation, problem in brief, proposed solution, resource requirements of the research, and structure of the thesis.

1.2 Aim and Objectives

This research aims to design and develop a panoptic segmentation approach to automatic floor plan analysis using vision transformers.

The following objectives have been identified to achieve the aim.

- Objective 1:- Critical review of methodologies used in automatic floor plan analysis.
- Objective 2:- In-depth study of technologies used in automatic floor plan analysis.
- Objective 3:- Design and development of a panoptic segmentation approach to automatic floor plan analysis using vision transformers.
- Objective 4:- Evaluate the developed solution using proper evaluation metrics.

1.3 Background and Motivation

Floor plan analysis encompasses various tasks and procedures, such as graphics separation, object detection and segmentation, structural modelling, and vectorization. The computer vision community has identified object detection and segmentation as the primary focus of floor plan analysis research. Rule-based and learning-based methods are widely utilized in floor plan analysis but have limitations. For example, rule-based algorithms hinge on specific plan styles, posing challenges for generalization to diverse forms. Conversely, learning-based models trained on distinct datasets exhibit adaptability. Yet, their outputs often lack precision due to pixel-level segmentation or substantial disparities in segmentation outcomes when objects are absent or small [8].

Object detection entails identifying and localizing objects in images. However, anchor-based and region-based approaches face challenges in recognizing objects under diverse conditions, mainly due to inadequate annotations and limited real-time performance [9]. With a small set of floor plan elements, object detection APIs like TensorFlow Object Detection API and earlier versions of YOLO have been experimented with, but they all included the above limitations. Semantic segmentation was formulated as a per-pixel classification task employing models like Fully Convolutional Network (FCN), U-Net, and DeepLab.

Meanwhile, instance segmentation is adopted with an alternate mask classification with models like Mask-Region Convolutional Neural Network (Mask-RCNN). It is important to note that all object detection approaches, as well as instance and semantic segmentation approaches, focus solely on individual tasks without considering the crucial need for a comprehensive floor plan analysis. This focus presents a constraint when comparing results, particularly when considering various metrics [8].

Addressing this, Kirillov et al. [10] introduced the concept of panoptic segmentation, which integrates both instance and semantic segmentation to yield a comprehensive understanding of the scene using the panoptic quality evaluation metric. Based on that, Cheng et al. proposed MaskFormer [11] and Mask2Former [12] models by indicating that mask classification is sufficiently general to solve semantic and instance segmentation by improving self-attention mechanisms in traditional vision transformers. However, these models require high-performance GPUs for training and datasets mapped with panoptic annotations. Fan et al. proposed CADTransformer [13] with pre-trained CNN to extract rich multi-resolution representations of floor plans for symbol spotting tasks in floor plans. However, it is a distinct task compared to this problem.

Aligned with the core area of floor plan research, this project was motivated by involvement in the construction sector called BuilderBid: Estimating Software for Builders & Contractors [14]. The cost estimation process necessitates manual measurement acquisition from floor plans. This involves segmenting room spaces, extracting line segments to depict room layouts, and tallying distinct objects like doors and windows. A significant observation is that when cost estimating, builders and contractors, on average, spend about 2/3 of their time just analyzing and measuring floor plans. Even with software assistance, this process is labor-intensive, requiring hours and sometimes days [4]. Hence, the need for an automatic procedure to comprehensively analyze floor plans is evident.

1.4 Problem in Brief

This research project directs its attention toward the extremely time-intensive, labor-intensive process of manual annotation and the inherent segmentation inaccuracies prevalent in conventional floor plan analysis, underlining the need for an automated solution to accelerate the process with a standard evaluation metric.

1.5 A Novel Approach for Automatic Floor Plan Analysis

The proposed approach employing panoptic segmentation through vision transformers can effectively revolutionize automatic floor plan analysis, which is the hypothesis of this research project. The input to the proposed solution consists of rasterized floor plan images, which serve as the primary data source for analysis. These images encompass diverse architectural drawings. The output of the proposed solution is a comprehensive analysis of the floor plan, including accurate segmentation of areas and objects, extraction of line segments, and identification of unique architectural elements.

The proposed solution introduces a panoptic segmentation approach, which aims to construct a comprehensive scene understanding by integrating object detection with image segmentation. This synergistic approach involves identifying and delineating individual object instances within those regions and subsequently segmenting images into meaningful regions. The methodology hinges on the efficacy of the Segment Anything Model (SAM) for segmentation, guided by input prompts generated through object detection models. Notably, the proposed approach diverges from the traditional practice of assigning semantic labels to individual pixels. Instead, it introduces a mask classification mechanism integrated within the segmentation stage.

The proposed solution is crafted to fulfil the needs of construction, architectural, interior design, and real estate professionals who require an automatic floor plan analysis process. Detects and segments object areas, such as rooms including room spaces, extracts line segments to capture the layout of rooms and recognize unique objects in the floor plan such as doors, windows, etc., are identified as the main features of the proposed solution.

1.6 Resource Requirements

On the hardware side, high-performance computing resources like GPUs are employed for training and inferencing the trained models. Programming languages like Python and Deep Learning frameworks like PyTorch are primarily adopted, along with available open-source floor plan datasets, to conduct experiments.

1.7 Structure of Thesis

The thesis has been structured in the following manner.

Chapter 1 offers a comprehensive introduction to the research project. It outlines the prolegomena, aim and objectives, background and motivation, research issues, and proposed solution.

Chapter 2 presents a literature review on automatic floor plan analysis, offering a comparative review of different procedures from rule-based to learning-based methods**.**

Chapter 3 provides a detailed description of the technologies implemented to establish the solution**.**

Chapter 4 presents the approach used to address the identified problems. This includes an explanation of the hypothesis, inputs, outputs, process, users, system requirements, and system features.

Chapter 5 focuses on the design of the proposed solution, covering conceptual design and modular design.

Chapter 6 describes the implementation of the proposed solution by the researcher.

Chapter 7 discusses the evaluation conducted on the established system.

Chapter 8 concludes the thesis by presenting the identified future opportunities and enhancements, as well as the conclusion and further work.

1.8 Summary

This chapter provides a comprehensive project overview, encompassing the research issue, aim and objectives, background, motivation, and innovative solution. The subsequent chapter will focus on a literature review of automatic floor plan analysis and the complexities in elucidating the research problem.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In Chapter 01, we presented an overview of the project. This chapter comprehensively analyses the research on advancements in automatic floor plan analysis. We have organized this chapter into distinct sections, including the evolution of automatic floor plan analysis from rule-based to learning-based methods, recent advancements in computer vision for floor plan analysis, the challenges associated with automatic floor plan analysis, research findings, and problem definition.

2.2 Evolution of Automatic Floor Plan Analysis

The following subtopics describe how automatic floor plan analysis evolved from rule-based to learning-based methods, including various techniques employed in each approach.

2.2.1 Rule-based Methods for Automatic Floor Plan Analysis

Early studies in analyzing floor plans primarily focused on object identification and modelling using CAD drawings, which provided accurate geometric information but lacked topological and semantic properties. Cherneff et al. [15] proposed methods for extracting the structural elements of floor plans, such as doors, rooms, walls, and windows, employing constrained drawing grammar. Shape Grammar (SG) emerged as a prevalent rule-based method, generating geometric shapes based on architectural styles. Additionally, initial studies included converting vector segments, interpreting hand-drawn plans, and recognizing symbols from hand-drawn sketches. However, the above studies did not consider the semantics and functional interactions between elements.

A semi-automatic room segmentation method was introduced by Ryall et al. [16] for raster plans, which utilized proximity metrics to identify regions. Although it had limitations in detecting false positives from various elements, it demonstrated the potential to recognize building semantics and constraints from a higher-level perspective. Automated generation of 3D layouts from analyzed floor plans involves tiling high-resolution images, segmenting pixels using morphological filtering, and

skeletonizing the segmented pixels for vectorized format and 3D reconstruction. Moreover, the advancement of rule-based techniques for detecting symbols, separating text, and vectorizing graphics revolutionized floor plan analysis by incorporating topological and semantic constraints. For one-story buildings, creating 3D models from rasterized floor plans includes manually detecting object symbols that are not directly associated with the plan's structure. Gimenez et al. [2] suggested a technique to create 3D models by detecting building elements based on structural rules. Figure 2.1 demonstrates the methodology for reconstructing a 3D building layout from rasterized floor plans.

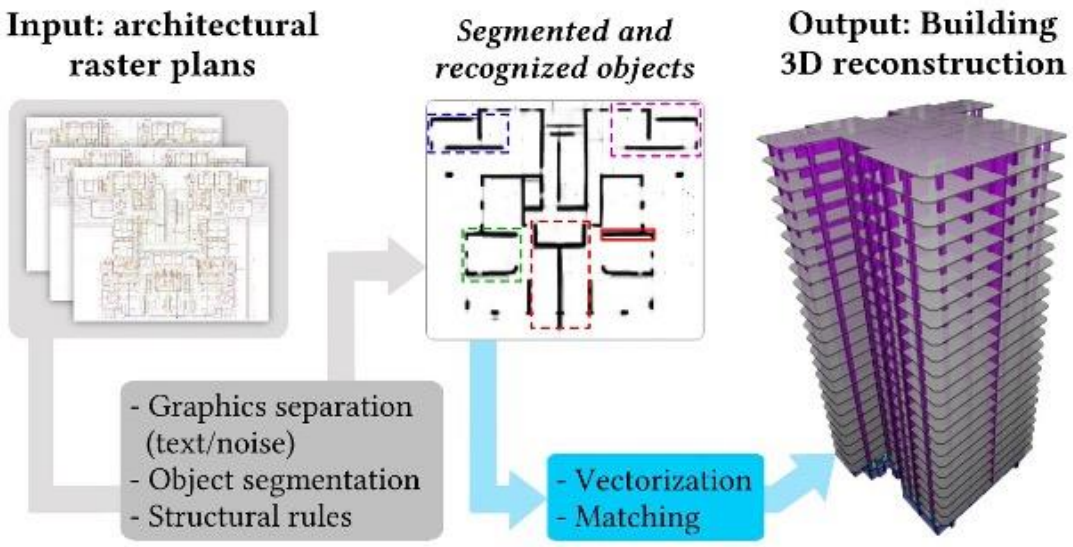


Fig. 2.1: Methodology for reconstructing a 3D building layout [2]

Mace et al. [5] developed an algorithm to detect rooms from scanned plans. However, these approaches are confined by predetermined parameters, manual pre-processing heuristics, or dependence on specific notations, limiting their generalizability. Other studies employed line representations to identify building components in floor plans. For example, Park and Kwon [6] identified primary walls utilizing auxiliary dimension lines. De las Heras et al. [17] presented an unsupervised wall segmentation method based on repetitive rectangular elements distributed across the plan. Nevertheless, these methods lack consideration of semantic relationships and require modification for different architectural designs.

Graph-based approaches have also been investigated to represent floor plan structures. A room layout segmentation algorithm was proposed using a Room Adjacency Graph (RAG) to identify rooms through furniture recognition in a graph-matching framework

[18]. Additionally, wall, door, and room identification using the Hough transform and polygonization. Although these methods capture connectivity between elements, they do not consider textual labels. Several techniques for low-level geometric vectorization have been devised for generic line drawings, but identifying floor layout components without accounting for their semantic relationships has presented a significant challenge. Notably, walls define the layout and constrain the placement of other elements.

2.2.2 Learning-based Methods for Automatic Floor Plan Analysis

Learning-based approaches have recently gained prominence to address the constraints of rule-based approaches. These models can directly discern complex relationships inherent in plan files from training data, amalgamating insights from architects, building codes, human creativity, and structural engineers. The development of Deep Learning (DL) models has propelled the field of automatic floor plan analysis forward, leading to notable breakthroughs. Various novel approaches have emerged to address the critical challenges in this domain, such as identifying wall joints to facilitate vectorization, employing generative image-to-image networks to unify plan formats, and integrating specialized loss functions to enhance structural reasoning during segmentation. These advancements have resulted in improved recognition metrics when compared to rule-based methods. Nevertheless, there remain obstacles that must be addressed.

2.2.3 Machine Learning in Automatic Floor Plan Analysis

Since 2017, there has been a notable surge in research focusing on learning-based methods for floor plan analysis. This increased interest can be ascribed to the accessibility of public datasets and advancements in computer vision models, particularly deep learning models. These learning-based approaches have shown promise in enhancing the accuracy and efficiency of floor plan analysis compared to traditional rule-based methods [6].

One of the early learning-based approaches initiates a grammatical model for interpreting plan documents [19]. Their approach utilized stochastic image grammar within an And-Or graph to depict building components' hierarchical, structural, and semantic relations. This work marked a significant step towards introducing a

trainable model for interpreting floor plans, although rule-based techniques were still required for identifying structural objects. To overcome the constraints associated with rule-based approaches, a style-agnostic, automatic method was suggested using a Support Vector Machine Bag of Visual Words (SVM-BOVW) [20]. This method sought to detect the pixel boundaries of structural elements by categorizing image patches into various types, such as doors, walls, and windows, from the BOVW model. The authors achieved improved results and style invariance, thus preventing the necessity for intricate ad-hoc rules for every notation.

A framework is proposed utilizing the α-shape algorithm for deriving shapes of rooms from binarized images and categorizing their characteristics [21]. Guo et al. [22] employed a pre-trained VGG-16 network to extract features, while a multi-layer perceptron was utilized for classifying wall shapes. Park and Kim [6] combined rule-based methods with the TensorFlow object detection API for 3D building modelling. Wall retrieval is achieved using a positive unlabeled learning-based approach. Following Figure 2.2 presents the VGG model architecture.

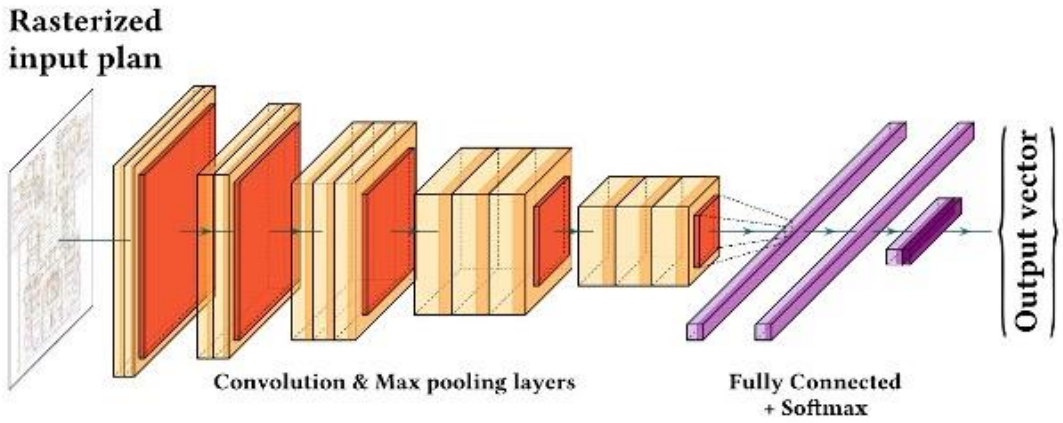


Fig. 2.2: VGG Model Architecture [22]

In conclusion, learning-based methodologies have greatly enhanced floor plan analysis by leveraging the capabilities of deep learning models and improving recognition accuracy and efficiency. These approaches have effectively eliminated the need for explicit rule-based methods in numerous instances and offer the potential to extend recognition across various floor plan styles. However, further research is required to overcome challenges and enhance the accuracy and generalizability of these methodologies.

2.2.4 Deep Learning in Automatic Floor Plan Analysis

Deep learning models, especially neural networks, have played a pivotal role in advancing floor plan analysis. These models have been utilized for various tasks, including object detection, instance segmentation, and semantic segmentation. They have enabled the extraction of spatial features, enhancing the recognition, classification, and vectorization of structural objects and rooms. However, challenges persist in effectively addressing noise and artifacts in segmented results and accurately recovering polygon shapes and contours.

Convolutional neural networks (CNNs) have been extensively employed in analyzing floor plans, with the automatic extraction of advanced features improving the recognition of structural objects [23]. Due to their ability to process two-dimensional tensors, such as image pixel matrices, CNNs are supervised learning algorithms widely utilized in computer vision tasks. The architecture of CNNs contains convolutional layers, non-linear processing units, and pooling layers. To transform the input data and highlight specific features in the output, convolutional layers use a convolution operator using kernel matrices (filters).

While it is possible to perform simple image processing tasks by manually defining kernel matrices, CNN models can produce more intricate and abstract features through training. The output of the convolutional layers is then passed through a non-linear processing unit (activation function), which enhances the network's abstraction capabilities and introduces non-linearity, facilitating the learning of semantic differences. A sampling layer, which summarizes the results and ensures interrelationship with geometrical distortions, can be used to follow the activation function output.

Although CNNs have demonstrated remarkable effectiveness in tasks such as image classification and segmentation, they have two primary limitations. Firstly, they lack interpretability, making it challenging for end-users to understand how the model works. Secondly, training CNNs necessitates a substantial number of labelled data to achieve accurate generalization.

In the domain of deep learning, models can be classified as either discriminative or generative-based. Discriminative models focus on learning the conditional probability distribution of classes, allowing for predictions on unseen data in assignments like

classification, regression, or segmentation. Their goal is to differentiate between different classes. Conversely, generative models learn the joint probability distribution, capturing the distribution of individual classes in a dataset to estimate probabilities for specific examples. Generative learning algorithms aim to capture the underlying patterns or distribution of data points and can generate new data points. Figure 2.3 depicts the deep learning models investigated in floor plan research, further detailed in the subsequent sections.

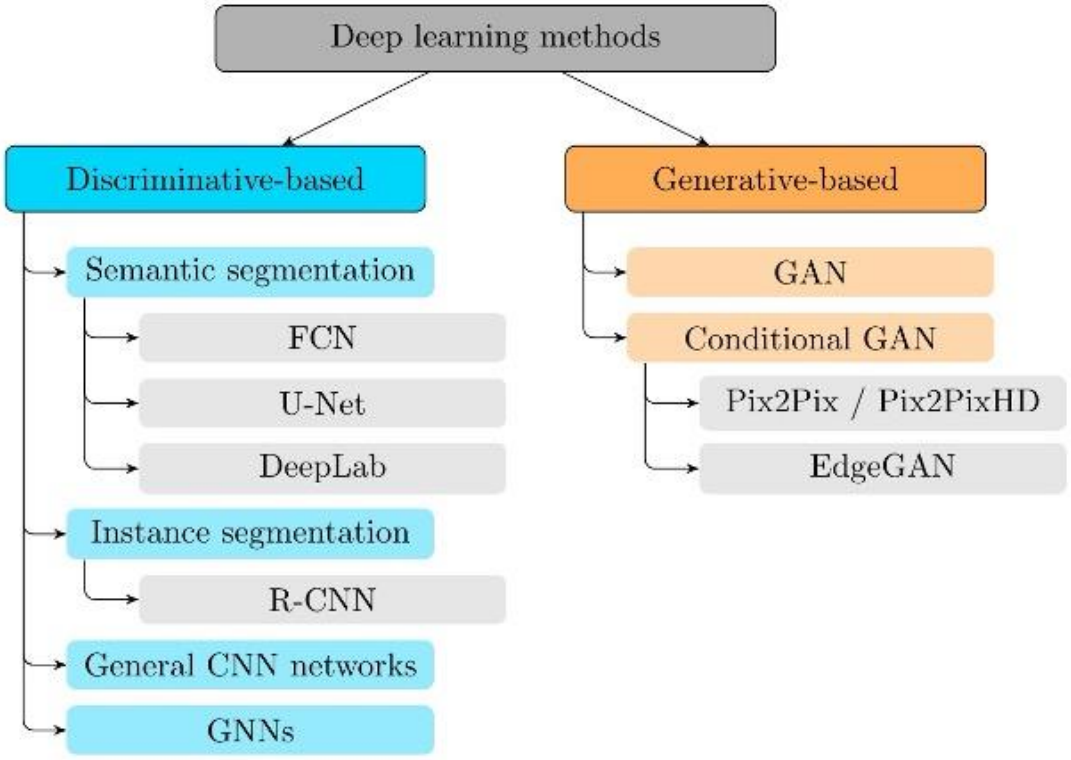


Fig. 2.3: Deep Learning methods investigated in studies [8]

2.2.5 Discriminative-based Models in Automatic Floor Plan Analysis

Discriminative-based models were widely used in floor plan analysis, including semantic segmentation models like FCN (Fully Convolutional Network), U-Net, and DeepLab, and instance segmentation models like RCNN (Region-based Convolutional Neural Network). FCNs consist of an encoder and a decoder. The encoder apprehends the image context through convolutional and max-pooling layers, while the decoder propagates context information to higher-resolution layers using transposed convolutions, resulting in segmented output. U-Net, similar to FCNs, employs an encoder-decoder architecture. The decoder enhances the localization and

reconstruction of the segmented output by integrating features and spatial information through up-convolutions and concatenations with high-resolution features from the encoder.

DeepLab [24] is a semantic segmentation model that utilizes a pre-trained CNN for extracting features and a decoder for reconstructing the segmented output. DeepLabV3+ with stacked convolutions has attained prominent results compared to its earlier versions. R-CNN [25] is an instance segmentation model that produces bounding boxes for each object. Faster R-CNN, a variant of R-CNN, is commonly used in floor plan analysis for object detection. Several studies have explored these models in floor plan analysis. For example, FCN-2s are used for wall segmentation, and Faster RCNN is used for object detection. In apartment floor plan analysis, FCN is used for semantic segmentation and graph modelling. U-Net [25] has also been applied for wall and door segmentation.

DeepLabV3+ has been widely used in floor plan analysis. DeepLabV3+ recognizes walls, windows, doors, and room types. Anchor-based instance segmentation models like Faster R-CNN (Faster Region-based Convolutional Neural Network) and YOLO (You Look Only Once) have been used to detect building elements such as walls, doors, and windows. Some studies have focused on capturing spatial characteristics to reconstruct floor plan elements. Liu et al. [26] used a CNN model to convert plans into vectorized formats by detecting wall junctions. Zeng et al. [27] introduced a deep multi-task neural network for segmenting room-boundary objects and recognizing room types. Figure 2.4 showcases the U-Net architecture.

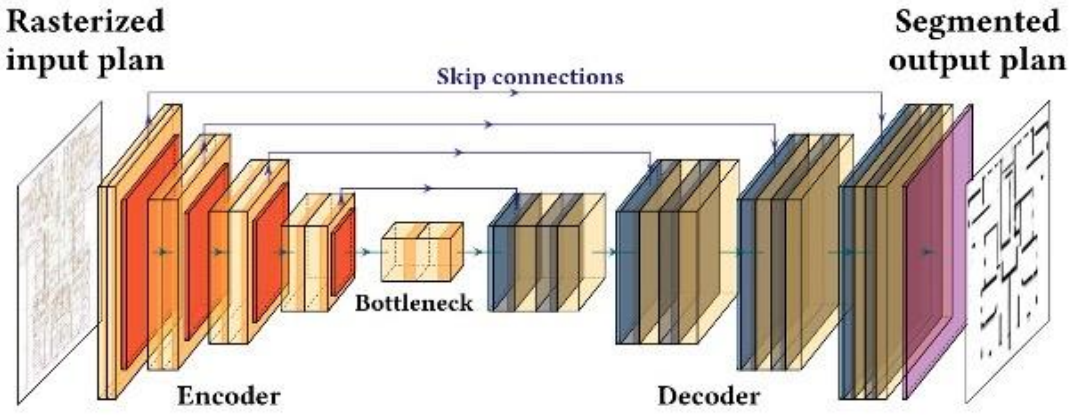


Fig. 2.4: U-Net model [25]

2.2.6 Generative-based Models in Automatic Floor Plan Analysis

Graph Neural Networks (GNN) have also been studied in the analysis of floor plans. Simonsen et al. [28] used GNN for door classification. A GNN-based framework for floor plan object vectorization was also developed. These models have shown promising results in various aspects of floor plan analysis, including segmentation, object detection, and vectorization. They offer improved performance compared to traditional methods and can handle different drawing styles and complexities.

Since the introduction of the Generative Adversarial Network (GAN) in 2014, notable advancements have been made in generative models and neural style transfer techniques. GANs, which consist of a generator and a discriminator, are trained with paired data to find optimal parameters that make it difficult for the discriminator to distinguish between generated and original data. GANs have evolved into various branches, including conditional GANs, Wasserstein GANs, and Pix2Pix, and have demonstrated success in tasks such as image translation, style transfer, noise reduction, super-resolution, image matting, semantic segmentation, and dataset augmentation.

One application of GANs is the recognition of structural objects. Zhang et al. [29] employed a multi-task GAN-based neural network with direction-aware, learnable, and additive kernels to improve the detection accuracy and segmentation of complex and irregular doors, rooms, walls, and windows. However, most researchers directed their attention towards employing GANs for image style transfer, enabling the extraction of primitives from intricate and overlapping graphics by harmonizing the level of detail from various types of drawings. Recent advancements in GANs, including conditional GANs (cGANs), CycleGANs, and DiscoGANs, have greatly improved image style transfer models.

Conditional GANs (cGANs) and CycleGANs [30] can transfer images across different styles while maintaining the fundamental structure. cGANs rely on labelled pairs in the dataset to guide the training process, while CycleGANs and DiscoGANs can learn from unpaired images using a two-cycle mapping approach. Although CycleGANs have broader applications and do not require pixel-level annotations, they face limitations in floor plan analysis because of the lack of extensive datasets, making conditional GANs the preferred choice in this context. Pix2Pix [31] is a GAN model, a significant milestone in image translation. It employs an encoder-decoder

architecture, such as the “U-Net” model, for the generator, enabling pixel-by-pixel mapping between two domains. Pix2Pix has been used for tasks such as generating a complete photo from a damaged one, converting black-and-white maps to colorful ones, and adding texture and shadow to linear sketches. Building upon Pix2Pix, Pix2PixHD expands its capabilities to handle high-resolution image synthesis and semantic manipulation. It introduces a novel adversarial learning objective and architecture for both generator and discriminator at multiple scales. Pix2PixHD is applied to detect and colorize rooms from floor plans, preserving the underlying structure while transforming the rasterized plan into a segmented style. Figure 2.5 shows the Pix2Pix model, which converts the style of a rasterized floor plan image into a segmented format.

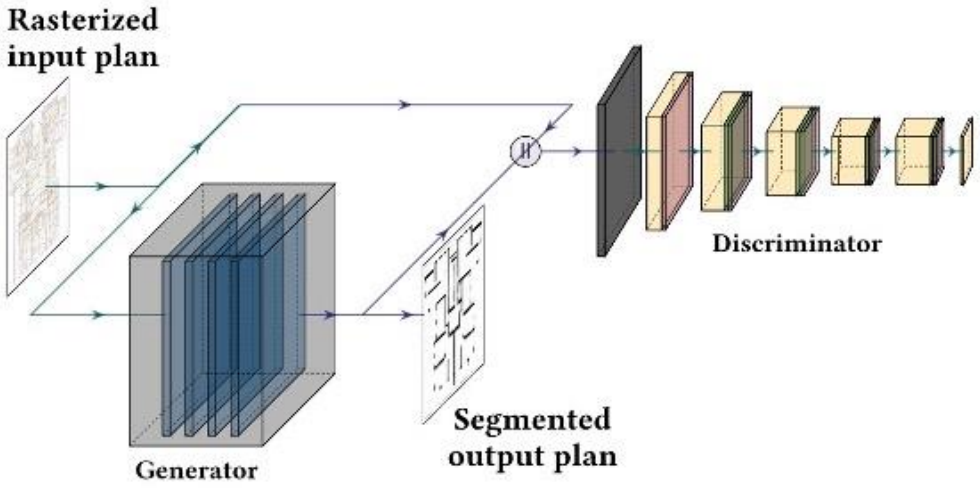


Fig. 2.5: Pix2Pix model [31]

Now, let’s explore recent advancements in computer vision to identify the potential technologies capable of mitigating the limitations encountered in current research on automatic floor plan analysis.

2.3 Recent Advancements in Computer Vision

The upcoming sections will delve into the latest developments in computer vision, specifically examining the Vision Transformers and Panoptic Segmentation. These state-of-the-art techniques will be explored for their potential use in automatic floor plan analysis.

2.3.1 Vision Transformers

Recently, remarkable advancements have been achieved in applying vision transformers [32] in various computer vision tasks, revolutionizing the field and pushing the boundaries of what is possible. Initially introduced for natural language processing, vision transformers have emerged as robust visual recognition and understanding models. One key aspect of their success is their ability to capture global context and long-range dependencies within images, enabling a holistic understanding of visual scenes. This global perspective is precious in object detection, semantic segmentation, and panoptic segmentation, where capturing spatial relationships and contextual information is crucial. Vision transformers have performed remarkably in these tasks, often outperforming traditional convolutional neural networks.

Moreover, vision transformers are highly scalable and can handle diverse architectural styles and designs, making them suitable for real-world applications. Additionally, the self-attention mechanism employed in vision transformers allows for effective information fusion across different models, enhancing overall performance. These recent advances in vision transformers have paved the way for exciting possibilities in computer vision, offering new avenues for exploration and innovation in areas such as autonomous driving, robotics, and medical imaging.

Vision transformers have recently demonstrated superiority over traditional convolutional neural networks (CNNs) in numerous computer vision tasks. One notable aspect where vision transformers outperform CNNs is their ability to capture global context and long-range dependencies within images. Unlike CNNs that rely on local receptive fields and hierarchical feature extraction, vision transformers leverage self-attention mechanisms to reason about relationships among tokens in the input image. This allows them to effectively model long-range dependencies and capture contextual information across the entire image. As a result, vision transformers have shown remarkable performance in tasks such as object detection, semantic segmentation, and panoptic segmentation.

Another advantage of vision transformers is their scalability and adaptability to diverse architectural styles and designs. CNNs typically require careful design and engineering to handle variations in image sizes or architectural structures. In contrast,

vision transformers are inherently more flexible, treating images as sequences of tokens, enabling seamless processing of inputs with different spatial resolutions or aspect ratios. This scalability makes vision transformers well-suited for real-world applications, where images can vary significantly in size, orientation, or layout.

Moreover, the self-attention mechanism employed in vision transformers allows for effective information fusion across different models. By attending to relevant features and their spatial relationships, vision transformers can better integrate information from multiple scales and levels of abstraction. This leads to enhanced overall performance, as the models can leverage local and global contextual cues to make more accurate predictions. This ability to effectively integrate information across the entire image contributes to the superior performance of vision transformers compared to CNNs in various computer vision tasks.

Two fundamental concepts have been instrumental in the advancement of conventional transformer models. The first concept, self-attention, addresses the challenge of capturing long-term dependencies among elements within a sequence. Unlike traditional recurrent models, which struggle to encode such relationships effectively, self-attention enables transformers to establish connections between distant elements in the sequence. This capability allows transformers to capture global context and dependencies, improving the understanding and representation of input data.

The second key idea revolves around pretraining on a large corpus of labelled or unlabeled data, followed by fine-tuning on a smaller labelled dataset specific to the target task. This pre-training procedure can be self-supervised, where the model learns to predict missing elements or generate plausible data examples. By pre-training on a vast amount of data, the model can learn valuable representations and acquire general knowledge about the underlying structure of the data. Subsequently, during the fine-tuning stage, the model adapts its expertise to the specific task using a smaller labelled dataset. This combination of pre-training and fine-tuning empowers transformers to leverage their learned representations effectively.

Recent advances in vision transformers have showcased their ability to surpass the performance of traditional CNNs in computer vision tasks. By leveraging global context, modelling long-range dependencies, and enabling effective information

fusion, vision transformers have opened up new possibilities and raised the bar for visual recognition and understanding. Figure 2.6 depicts the current cutting-edge vision transformers in computer vision.

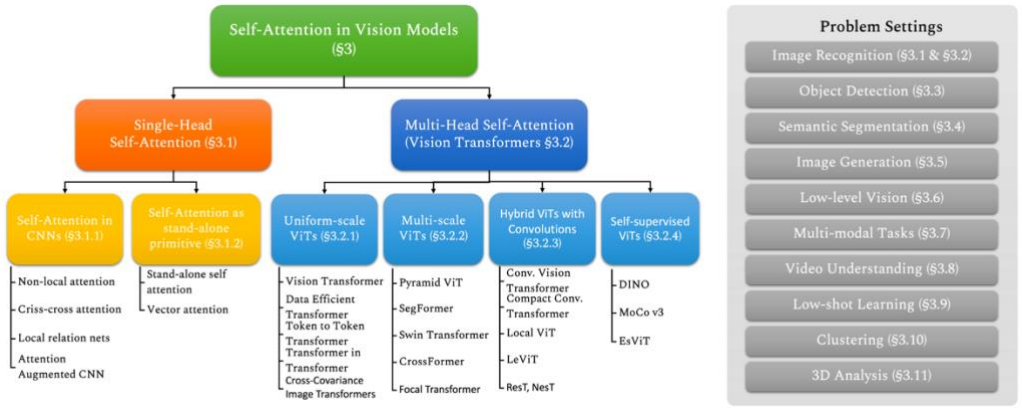


Fig. 2.6: Current sota vision transformers in computer vision [32]

2.3.2 Panoptic Segmentation

Panoptic segmentation [10] is an advanced computer vision task that aims to simultaneously classify and segment all objects within an image, including both "stuff" (like grass, road, sky) and "things" (like cars, people, trees). Unlike traditional instance segmentation, which focuses on identifying and delineating individual objects, panoptic segmentation provides a holistic understanding of the scene by assigning unique labels to semantic regions and specific instances. This task requires the model to comprehend the global context of the image while accurately segmenting each object instance.

Panoptic segmentation algorithms typically combine convolutional neural networks (CNNs) and transformer architectures to achieve superior results. By combining the spatial awareness of CNNs with the long-range relationship modelling of transformers, panoptic segmentation algorithms can effectively handle complex scenes with diverse object categories, sizes, and occlusions. The development of panoptic segmentation techniques has opened up new possibilities for scene understanding, enabling applications in autonomous driving, robotics, augmented reality, and more. With ongoing research and advancements in computer vision, panoptic segmentation continues to evolve, pushing the boundaries of what is possible in scene analysis and interpretation.

According to [10], semantic, instance, and panoptic segmentation are three essential computer vision tasks that analyse visual data at a pixel level. Semantic segmentation assigns semantic labels to pixels, providing a detailed understanding of the scene's composition. Instance segmentation goes further by differentiating between individual object instances, enabling precise localization and distinction between objects. Panoptic segmentation combines semantic and instance segmentation, partitioning the image into meaningful regions, including objects and background. Evaluation of these tasks involves different metrics. Semantic segmentation utilizes IoU, dice coefficient, pixel accuracy, and mean accuracy. Instance segmentation uses Average Precision (AP) based on pixel-level IoU. Panoptic segmentation employs Panoptic Quality (PQ), combining segmentation quality (SQ) and recognition quality (RQ). SQ measures average IoU, and RQ is the F1 score of predicted masks. These metrics enable accurate performance assessment and drive advanced computer vision techniques for detailed visual data analysis. Figure 2.7 represents the panoptic segmentation example scenario.

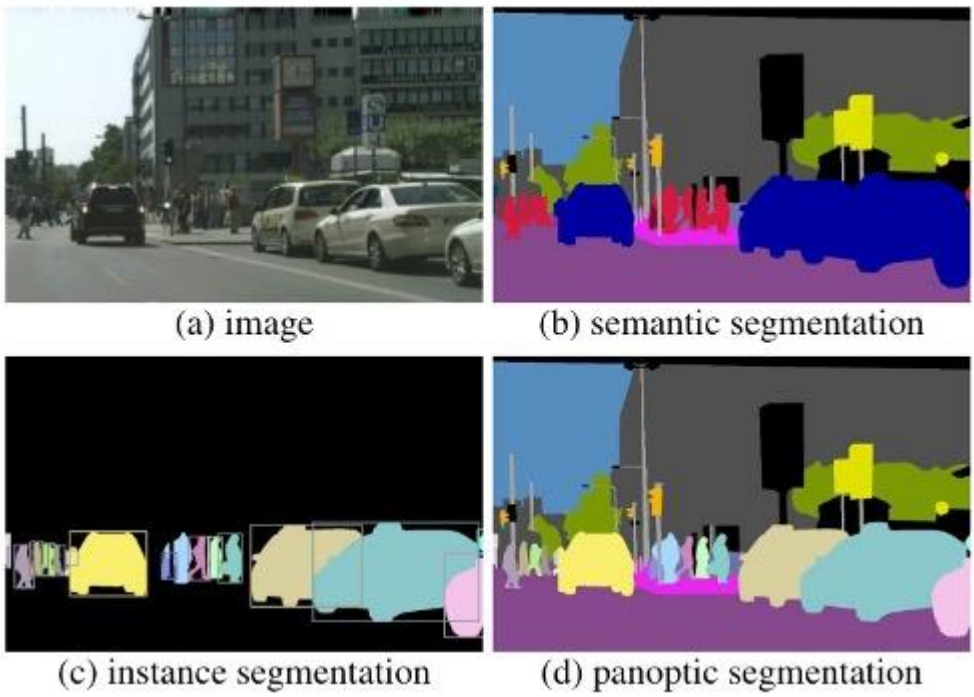


Fig. 2.6: Panoptic segmentation [10]

Next, let's delve into the persistent challenges faced in the automated analysis of floor plans.

2.4 Challenges in Automatic Floor Plan Analysis

One major challenge is the absence of standardization in result analysis [8]. Each article in the field uses different evaluation metrics, making it difficult to compare results across studies. Metrics such as the detection rate, jaccard index, intersection over union, mean average precision, match score, recognition accuracy, pixel/class accuracy, precision, recall, and confusion matrix have been used. Still, no consensus exists on which metrics should be used for specific tasks.

Furthermore, the absence of shared annotations for complex floor plan datasets poses a barrier to comparing learning-based approaches. Standardizing analysis procedures and adopting a standard metric would enable better comparison and selection of models based on plan style and task requirements. The availability of new public datasets is another challenge. Current datasets primarily focus on houses and apartments, limiting the diversity of plan styles and types. Expanding the dataset scope to include office layouts, apartment plans, public structures, and more intricate examples would better represent the industry's needs and allow for broader analysis from various sources.

Annotating floor plans is costly and challenging, mainly due to the absence of standardized notation and ambiguous situations [8]. Unsupervised DL models offer a solution by allowing analysis without annotated floor plans. Although some studies have suggested unsupervised approaches, they frequently depend on stringent assumptions and intricate learning criteria. Developing non-supervised models that can learn relationships and structural reasoning would enable the recognition of complex objects in novel plan designs and facilitate the examination of unexplored datasets. Integrating rule-based and learning-based methodologies can offer a comprehensive approach.

Learning-based algorithms excel in tasks requiring generalization, while rule-based models handle fine details and deterministic responses. Leveraging both mechanisms allows for comprehensive floor plan analysis, addressing challenges that are difficult for either approach individually. The industry also has a growing demand for automated floor plan analysis. Architectural firms and engineering offices can benefit from algorithms that can process and analyze floor plans in batches, automating tasks

such as recognition, vectorization, modelling, and searching in large databases. Meeting these industry needs can drive further advancements in the field.

In the following section, let's dive into the existing datasets to analyze floor plans.

2.5 Existing Datasets

Datasets have been instrumental in floor plan analysis, particularly without a standardized notation for their composition. Consequently, models developed for floor plan analysis must integrate tailored rules for each style [8]. Implementations in this domain encounter significant design variability stemming from three primary factors. Deviations from standard regulations are initially common in plan representations, with approximately 70% of the graphical information adhering to norms. Moreover, the inherent flexibility of floor plan documents permits virtually limitless configurations and relationships among plan elements. Furthermore, each dataset of floor plans presents limitations in terms of quantity or complexity, prompting researchers to select datasets aligning with their specific objectives, potentially necessitating customized processing procedures.

Annotating floor plans with objects like rooms and walls is essential for their utility in floor plan analysis. However, this task is intricate and costly due to the expertise needed to identify various elements and the ambiguity in notation [8]. For instance, windows in specific plans may overlap with the beams, and slabs might include paths, shafts, or custom symbols specified by architectural and structural firms. Despite developing practical annotation tools to simplify the process, ensuring consistent annotations across different experts remains challenging, especially for complex plans.

As floor plans are typically proprietary assets belonging to individuals or companies, their availability is often restricted. Therefore, we explored open-source floor plan datasets to conduct experiments for this research project. We meticulously examined datasets containing floor plan documents that closely resemble real-world scenarios.

2.6 Research Findings

Table 2.1 compares research findings, including the limitations and drawbacks of rule-based and learning-based methods and how vision transformer-based models can be adopted to tackle those issues.

TABLE 2.1: COMPARISON OF RESEARCH FINDINGS

Method	Limitations and Drawbacks	Vision Transformer Models
Rule-Based Methods	Manual rule creation is time-consuming and labor-intensive.	Automate feature extraction, eliminating the need for manual rule creation.
	Difficult to handle complex and non-linear patterns.	Capture complex patterns by leveraging the self-attention mechanism, allowing them to model long-range dependencies in the data.
	Prone to errors and brittleness when dealing with noisy or ambiguous data.	They can learn from large-scale datasets, making them more robust to noisy and ambiguous data.
	Lack of generalization ability to handle unseen data.	Learning rich representations from diverse visual data enables better generalization of unseen data.
Learning-Based Methods	Needs a substantial amount of labelled training data.	Can leverage large-scale labelled datasets for pre-training, enabling them to learn rich visual representations without requiring as much labelled data.

	Prone to overfitting or underfitting if the training data is not representative.	Can reap advantages from transfer learning by initially training on extensive datasets and optimizing for specialized tasks, reducing the risk of overfitting with limited training data.
	Difficulty in interpreting and explaining decisions.	Have a self-attention mechanism that enables interpretability by allowing visualizations of the attended regions, providing insights into the model's decision-making process.
	Not inherently capable of handling sequential data.	Incorporate positional encodings, allowing them to capture the spatial relationships in the image.

2.7 Problem Definition

Having considered the summary of limitations/drawbacks identified in automatic floor plan analysis, we have noticed that most of the researchers failed to provide an in-depth analysis of a floor plan, which should cover a vast amount of information by giving room-by-room information (unique objects in the room, area of the room, wall lines of the room) of a floor plan. We also notice many researchers have used almost the same approach across the rule-based and learning-based methods, but they have many limitations and drawbacks. We also gather ample evidence to prove that vision transformers can effectively address the limitations and drawbacks we identified from the existing research. Additionally, the integration of these efforts into production-ready software remains unexplored. Based on our critical review, we define our research problem towards the extremely time-intensive, labor-intensive process of manual annotation and the inherent segmentation inaccuracies prevalent in conventional floor plan analysis, underlining the need for an automated solution for accelerating the process with a standard evaluation metric.

2.8 Summary

In this chapter, we have critically reviewed automatic floor plan analysis by highlighting techniques used in rule-based and learning-based methodologies with the limitations/drawbacks of those approaches. We have also identified various technologies used for automatic floor plan analysis. More importantly, we have defined our research problem towards the extremely time-intensive, labor-intensive process of manual annotation and the inherent segmentation inaccuracies prevalent in conventional floor plan analysis, underlining the need for an automated solution for accelerating the process with a standard evaluation metric. The literature review identified vision transformers with panoptic segmentation as a potential automatic floor plan analysis technology. Chapter 03 discusses the technology we adopt for automatic floor plan analysis.

CHAPTER 3

TECHNOLOGIES ADOPTED

3.1 Introduction

Chapter 02 delves into an extensive analysis of the existing literature concerning methods utilized in automatic floor plan analysis. The review methodically delineates the research hurdles and scrutinizes various proposed technological strategies. Consequently, Chapter 03 provides detailed insights into the technological framework adopted in our solution for automatic floor plan analysis, which integrates panoptic segmentation alongside object detection and promptable segmentation as the principal techniques, establishing a robust basis for addressing the intricacies inherent in floor plan analysis.

3.2 Computer Vision

Within computer vision, core concepts encompass object detection, instance segmentation, semantic segmentation, and panoptic segmentation, all pivotal for holistic scene comprehension. Object detection entails recognizing and pinpointing individual objects in an image with bounding boxes. Instance segmentation expands upon this by distinguishing separate instances of objects and assigning unique segmentation masks to each. Semantic segmentation classifies pixels into specific categories, providing detailed labelling based on semantic meaning. Panoptic segmentation integrates instance and semantic segmentation to offer a unified understanding of stuff and things within an image, facilitating comprehensive scene analysis.

Conventionally, achieving panoptic segmentation in floor plans requires training models on specialized datasets with panoptic annotations, which can be a significant hurdle. However, a recent breakthrough from Meta AI, the Segment Anything Model (SAM), offers a game-changing solution. SAM’s versatility lies in its ability to handle diverse segmentation tasks effectively within a single, unified framework. This removes the need for complex, multi-model approaches. SAM’s unique prompt-based segmentation also allows for greater control over the generation of segmentation masks, providing a more adaptable and user-friendly solution.

3.2.1 Segment Anything

Segment Anything Model (SAM) [33], developed by Meta AI, is an innovative image segmentation model designed for versatility and ease of use. Unlike traditional models that require extensive training for specific tasks, SAM simplifies the process by acting as a foundational model. SAM offers a streamlined method for panoptic segmentation using a single model, unlike conventional models, which leverage an ensemble of multiple networks. It utilizes a transformer-based design to predict semantic classes and instance IDs per pixel through an attention-based sequential grouping algorithm. SAM boasts several key features that set it apart:

- Prompt-based segmentation:- SAM utilizes user prompts (clicks, bounding boxes, text descriptions) to identify the desired segmentation target, eliminating the need for large, task-specific datasets for training.
- Zero-shot learning:- SAM excels at transferring its knowledge to new image distributions and tasks without requiring additional training on those specific cases.
- Multiple segmentation modes:- Users can segment all objects in an image or specific ones based on prompts, enhancing usability.

Figure 3.1 represents the overview of the Segment Anything Model architecture.

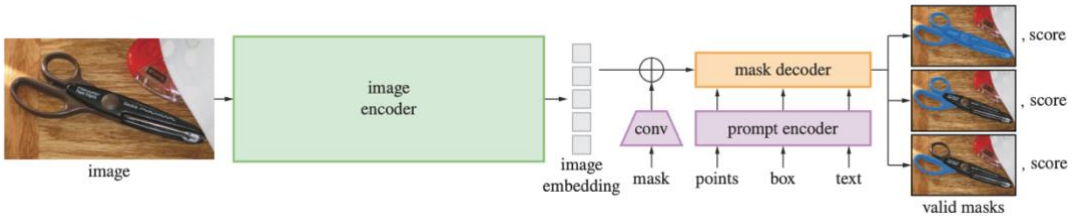


Fig. 3.1: Segment Anything Model [33]

SAM's architecture is built on three key components:

- Image Encoder:- Analyzes the image and creates a digital representation capturing its essential features.
- Prompt Encoder:- Interprets the user's prompt, whether text or visual and translates it into a format the model understands.
- Mask Decoder:- Utilizes the information from the image and prompt encoders to generate a precise segmentation mask, outlining the desired objects.

3.2.2 You Look Only Once (YOLO)

The YOLO (You Only Look Once) framework presents a neural network architecture that predicts class probabilities and bounding boxes in a single end-to-end model [34]. This constraints conventional object identification methods, often repurposing classifiers for detection purposes. By taking a distinct methodology in object detection, YOLO has demonstrated exceptional performance, surpassing other real-time object detection algorithms by a significant margin. Unlike methods such as Faster R-CNN, which employ a Region Proposal Network to identify potential regions of interest before performing recognition tasks separately, YOLO streamlines the process by conducting all predictions through a single fully connected layer. Furthermore, while methodologies utilizing Region Proposal Networks necessitate multiple iterations for the same image, YOLO efficiently achieves the task with a single iteration.

The YOLO algorithm utilizes a deep convolutional neural network (CNN) to identify objects in input images. The CNN architecture entails the following key components:

- Pre-training with ImageNet:- The model's first 20 convolution layers are trained using the ImageNet dataset during the initial pre-training phase. This stage includes temporary average pooling and fully connected layers.
- Grid-based Division:- YOLO partitions the input image into an S × S grid. Each grid cell detects objects if their centers lie within the cell boundaries.
- Bounding Box Prediction:- In every grid cell, the model makes predictions for bounding boxes along their corresponding confidence scores. The confidence scores indicate the model's level of certainty regarding the presence of an object within a box and the accuracy of the box prediction.
- Specialization of Bounding Box Predictors:- YOLO proposes several bounding boxes for each image area but allocates only one predictor per object during training. This chosen box has the greatest Intersection over Union (IOU) with the object's actual location.
- Non-Maximum Suppression (NMS):- When multiple boxes are generated, NMS cleans up these extras, keeping only the most likely bounding box for each object. This ensures a single, clear box for each object in the final results.

3.2.3 YOLOv8

YOLOv8, the latest iteration of the YOLO model developed by Ultralytics, represents the forefront of object detection technology [35]. Continuously evolving from its predecessors, YOLOv8 introduces new features and enhancements to deliver superior performance, adaptability, and efficiency. Distinguished as a state-of-the-art (SOTA) model, YOLOv8 caters to a comprehensive spectrum of vision AI tasks, including detection, segmentation, pose estimation, tracking, and classification. This extensive functionality empowers users to harness YOLOv8's capabilities across various applications and domains. Through its advanced architecture and versatile capabilities, YOLOv8 stands poised as a formidable solution for addressing diverse challenges in computer vision with unparalleled precision and effectiveness.

YOLOv8's architecture is built on the following key features:

- Advanced Backbone and Neck Architectures:- YOLOv8 integrates state-of-the-art backbone and neck architectures to enhance feature extraction and boost object detection performance significantly. An updated version of the CSPDarknet53 architecture shapes the backbone of YOLOv8.
- Anchor-free Split Ultralytics Head:- YOLOv8 adopts an anchor-free split Ultralytics head, departing from conventional anchor-based methods. This innovative approach improves accuracy and streamlines the detection process for enhanced efficiency.
- Self-Attention Mechanism:- YOLOv8 incorporates a self-attention mechanism within its network architecture, enabling dynamic prioritization of different image regions based on their relevance to the task.
- Multi-Scaled Object Detection:- YOLOv8 excels in multi-scaled object detection by utilizing a feature pyramid network, which comprises multiple layers dedicated to detecting objects of varying sizes and scales within an image. By leveraging this architecture, YOLOv8 can effectively identify large and small objects, enhancing its overall detection capabilities.

The following Figure 3.2 illustrates the detailed visualizations of YOLOv8 architecture.

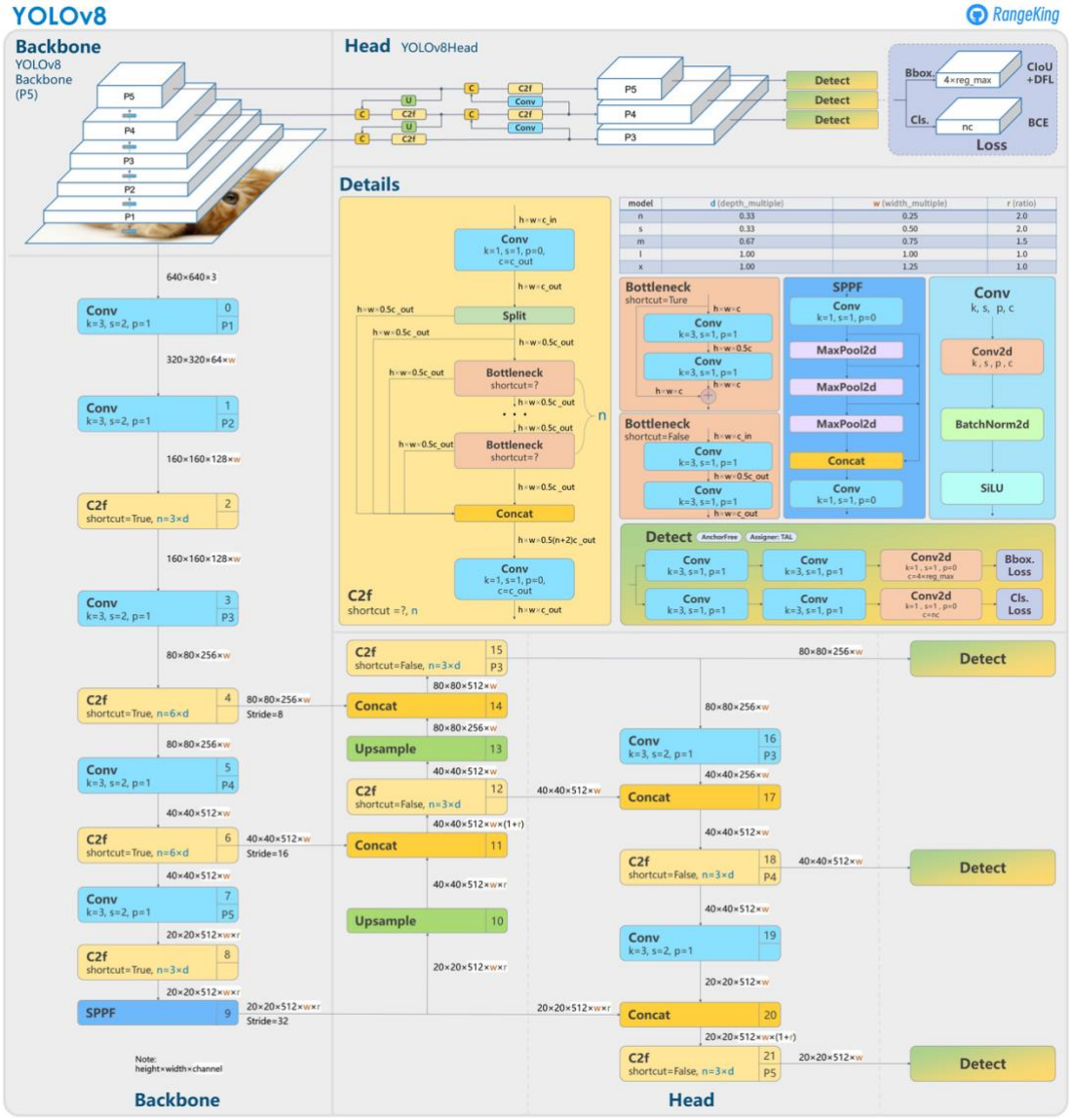


Fig. 3.2: YOLOv8 Architecture [35]

3.2.4 Real-Time Detection Transformer (RT-DETR)

Real-Time Detection Transformer (RT-DETR), crafted by Baidu, is a state-of-the-art end-to-end object detector renowned for its real-time capabilities and exceptional accuracy [9]. It leverages Vision Transformers (ViTs) to adeptly handle multiscale features by distinguishing between intra-scale interaction and cross-scale fusion. Offering remarkable adaptability, RT-DETR facilitates seamless adjustment of inference speed through various decoder layers without necessitating retraining. The model demonstrates unparalleled performance on accelerated backends such as CUDA with TensorRT, surpassing numerous counterparts in real-time object detection.

The following Figure 3.3 represents the RT-DETR architecture.

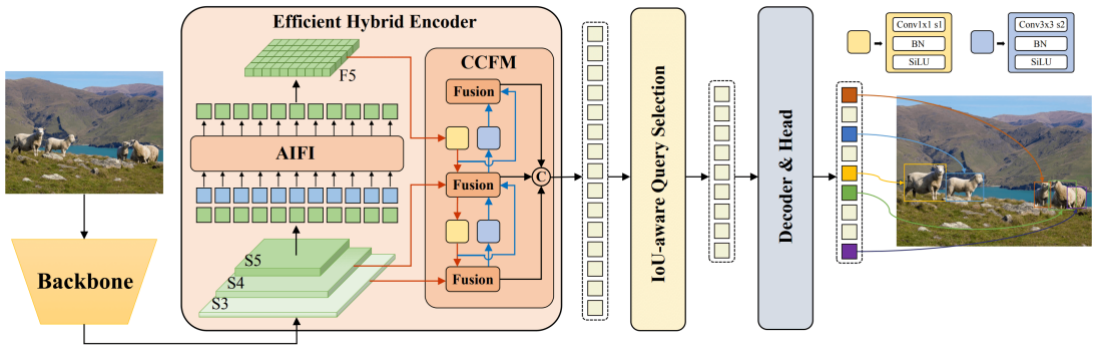


Fig. 3.3: RT-DETR Architecture [9]

The RT-DETR model architecture diagram illustrates that the final three stages of the backbone {S3, S4, S5} serve as input to the encoder. This efficient hybrid encoder converts multiscale features into a sequence of image features by employing intrascale feature interaction (AIFI) and a cross-scale feature-fusion module (CCFM). The architecture implements IoU-aware query selection to select initial object queries for the decoder. This process involves choosing a predetermined number of image features with the highest potential to represent objects accurately. Subsequently, the decoder and auxiliary prediction head iteratively refine these initial object queries to generate bounding boxes and confidence scores. Through the iterative refinement process, object queries are optimized, resulting in improved accuracy and reliability of the final detection outcomes.

3.3 Roboflow

Roboflow [36] was elected as the primary research project platform. With its comprehensive suite of features, Roboflow offers an all-in-one solution for managing and deploying computer vision applications efficiently. From data management to annotation, model training on powerful GPUs, and seamless deployment across diverse environments, Roboflow covers every aspect of the computer vision pipeline. Roboflow offers open-source options and an intuitive hosted platform, ensuring easy adaptation and customization. With features such as data preprocessing, collaborative annotation, and support for various formats and deployment environments, Roboflow empowers us to realize our vision efficiently.

The following Figure 3.4 shows the overview of the Roboflow platform.

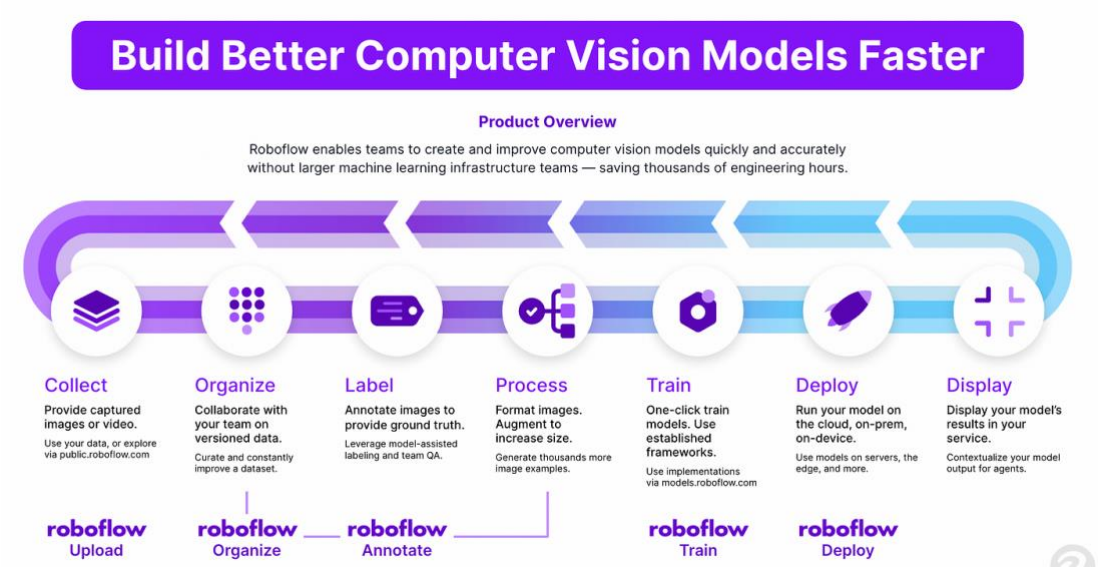


Fig. 3.4: Roboflow Product Overview [36]

3.4 Programming Languages

Python was elected as the main programming language for developing the system, selected for its flexibility, user-friendliness, and robust support for computer vision and ML/DL libraries. Its extensive array of libraries and tools for data processing, analysis, and visualization proved invaluable throughout the system's development. Python's straightforward syntax and potent capabilities rendered it ideal for implementing various AI-related models and algorithms. Moreover, its large community and wealth of resources facilitated troubleshooting and optimization efforts, enhancing the overall development process.

3.5 Machine Learning Frameworks

Machine learning frameworks are comprehensive toolsets explicitly designed for developing, training, and deploying ML models. These frameworks offer pre-written code libraries encompassing many algorithms and tools. By leveraging these pre-built components, developers can focus on the unique aspects of their projects, eliminating the need to write code from scratch.

3.5.1 PyTorch

This research project leverages PyTorch, a freely available machine learning toolkit pioneered by Meta AI.

Below are the core elements and functionalities of PyTorch:

- Intuitive Pythonic API:- PyTorch leverages the familiar Python syntax, resulting in a user-friendly and readable codebase. This intuitive design lowers the barrier to entry for developers, especially those already comfortable with Python.
- Dynamic Computational Graph:- Unlike some frameworks with static computational graphs, PyTorch utilizes a dynamic approach. This allows for greater flexibility during model development, enabling developers to define the computational graph on the fly. This dynamism empowers rapid experimentation and exploration of different model architectures.
- Eager Execution:- PyTorch employs eager execution, where operations are executed immediately upon definition. This approach facilitates debugging and understanding the model's behavior as each line of code is executed. This can be particularly beneficial for rapid prototyping and troubleshooting.
- Automatic Differentiation:- A cornerstone of deep learning, automatic differentiation allows for efficient gradient calculation, which is crucial for training models. PyTorch seamlessly handles this process, enabling developers to focus on model design rather than manual gradient calculations.
- Rich Ecosystem:- PyTorch boasts a thriving ecosystem with a large and active community. This translates to extensive documentation, tutorials, and online forums for troubleshooting, knowledge sharing, and staying updated on the latest advancements. PyTorch integrates seamlessly with the broader Python scientific computing ecosystem, including libraries like NumPy and SciPy.

3.6 Cloud Computing

Google Colab was established as the cloud-based platform for executing Python code within a web browser. While the free tier provides a valuable entry point for many users, Google Colab Pro unlocks a comprehensive suite of advanced features designed to streamline and accelerate the development of Machine Learning projects. Core functionalities of Google Colab Pro include extended runtime sessions, accelerated hardware resources, background execution, prioritized access, and increased computer units.

3.7 User Interface

The user interface for the system was built using Streamlit [37], which departs from the conventional web development paradigm that relies on writing code in HTML, CSS, and Javascript. Instead, it embraces a declarative approach for data scientists and machine learning engineers. This approach describes the desired user interface elements and their functionalities within Python code. Streamlit then handles the underlying technical details, translating that code into a functional and visually appealing web application.

3.8 Summary

This chapter delves into the technological foundation of our research project. We begin by highlighting concepts of computer vision and models mainly adopted, such as Segment Anything, YOLOv8, RT-DETR, etc. We select Roboflow as the primary platform to build end-to-end computer vision models. Following this, we discuss the core Machine Learning framework, PyTorch, that facilitated model development alongside the integration of cloud computing for efficient training and fine-tuning of the models. Finally, we briefly highlight the technologies to develop the final system's interface. The strategic selection of these technologies played a pivotal role in forming a precise and efficient system designed explicitly for automatic floor plan analysis. This optimized technological foundation significantly contributed to the project’s success.

CHAPTER 4

A NOVEL APPROACH FOR AUTOMATIC FLOOR PLAN ANALYSIS

4.1 Introduction

The chosen technological framework for automatic floor plan analysis is examined in the prior chapter. This chapter outlines the approach the researcher uses to drive towards a solution.

4.2 Hypothesis

The researcher hypothesizes that the proposed panoptic segmentation approach using vision transformers for floor plan analysis will be an effective solution compared to traditional supervised and unsupervised methods. By leveraging the power of transformer architectures, the framework will effectively capture spatial distribution, line segments, and object details, enabling comprehensive floor plan analysis. The integration of visual prompting will leverage their respective strengths, leading to improved segmentation results and reduced reliance on manual annotation.

4.3 Input

The input to the proposed solution is a rasterized floor plan image, which serves as the primary data source for analysis. The image may contain complex drawings, varying annotations, and the absence of standardized notations.

4.4 Output

The output of the proposed solution is a comprehensive analysis of the floor plan, including accurate segmentation of areas and objects, extraction of line segments, and identification of unique architectural elements. The output will provide insights into the floor plan's spatial distribution, connectivity, and functional components.

4.5 Process

The proposed solution introduces a panoptic segmentation approach, which aims to construct a comprehensive scene understanding by integrating object detection with image segmentation. This synergistic approach involves identifying and delineating individual object instances within those regions and subsequently segmenting images into meaningful regions. The methodology hinges on the efficacy of the Segment Anything Model for segmentation, guided by input prompts generated through object detection models. Notably, the proposed approach diverges from the traditional practice of assigning semantic labels to individual pixels. Instead, it introduces a mask classification mechanism integrated within the segmentation stage.

This approach enables the models to capture underlying patterns and structures from unlabeled floor plan images. By integrating visual prompting techniques, the analysis of floor plans is enhanced, providing accurate segmentation, precise identification of line segments and objects, and a comprehensive understanding of the spatial distribution and architectural components of floor plans. This innovative approach revolutionizes floor plan analysis, empowering designers, architects, and stakeholders with improved decision-making capabilities and increased productivity.

The proposed solution for automatic floor plan analysis can be illustrated below in Figure 4.1.

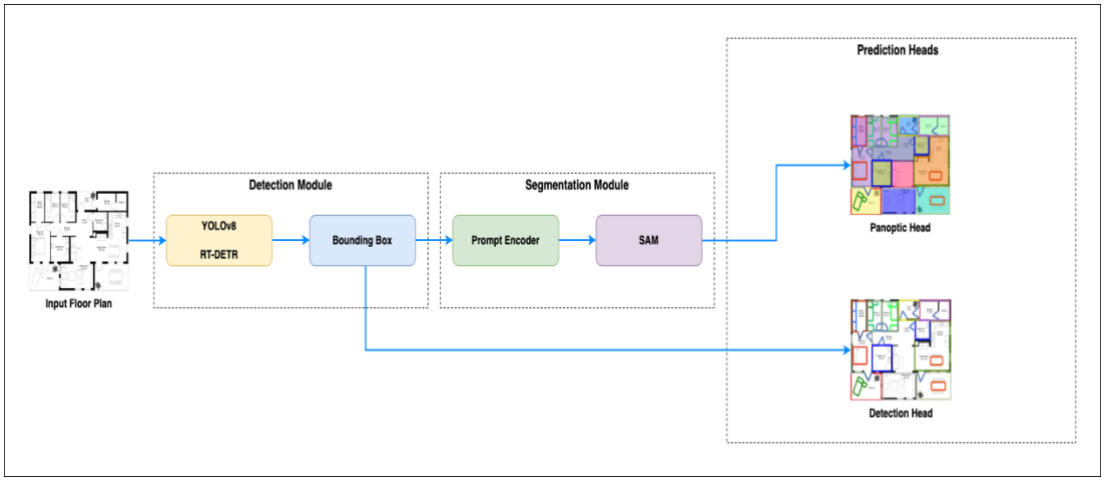


Fig. 4.1: Pipeline of our approach prompt input generated from object detection models and combining SAM model for segmentation

4.6 Features

The overall features of the system include the following.

- Detects and segments object areas, such as rooms, including room spaces.
- Extracts line segments to capture the layout of rooms.
- Recognizes unique objects in the floor plan, such as doors, windows, etc.
- Enables users to select different models (for properties, room spaces, and furniture types) and generate results according to their needs.
- Provides performance evaluation metrics to assess segmentation and recognition quality.

4.7 Users

The proposed solution is tailored to meet the needs of professionals in the architectural, interior design, and real estate industries who require precise floor plan analysis.

4.8 Summary

In this chapter, the hypothesis, input, output, features, users, and process of the proposed solution are thoroughly examined. The researcher elucidates the input pertinent to the proposed solution, defining the method and procedures employed to transform inputs into outputs. Furthermore, the critical features of the BuilderFormer were addressed towards the conclusion of this chapter. The subsequent chapter will introduce the concepts of the suggested solution.

CHAPTER 5

DESIGN OF BUILDERFORMER

5.1 Introduction

Chapter 05 delves into the design of the BuilderFormer, delineating its structure across five key modules. Each module's fundamental functions are elucidated alongside exploring their interplay within the system.

5.2 High-Level Design of the System

The input to the system is a rasterized floor plan image, which serves as the primary data source for analysis. The image may contain complex drawings, varying annotations, and the absence of standardized notations. The system's output is a comprehensive floor plan analysis, including accurate segmentation of areas and objects, extraction of line segments, and identification of unique architectural elements. The output will provide insights into the floor plan's spatial distribution, connectivity, and functional components. The process followed in converting the inputs to outputs is called a panoptic segmentation approach, which aims to construct a comprehensive scene understanding by integrating object detection with image segmentation with specialized models to enhance the detection and segmentation of floor plans. BuilderFormer is composed of five modules, namely, pre-processing module, detection module, post-processing module, segmentation module, and inference module. Figure 5.1 presents the conceptual design of the system.

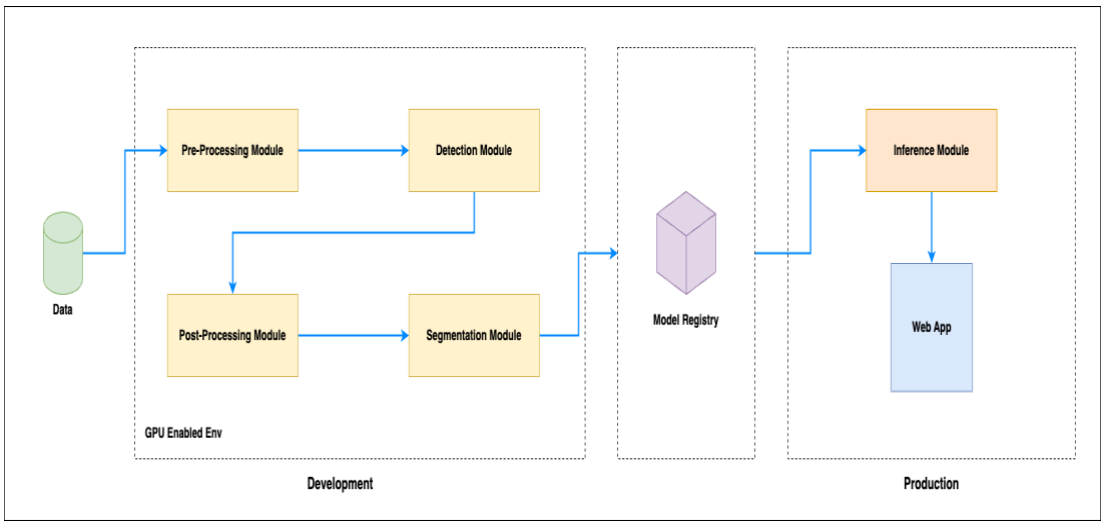


Fig. 5.1: Conceptual design of the system

5.3 Floor Plan Data Preparation

Several datasets have been identified within the realm of automatic floor plan analysis, each serving as valuable resources for research, development, and evaluation. These datasets offer diverse collections of floor plans, facilitating comprehensive exploration and analysis within the domain. Table 5.1 below represents the datasets that were chosen for this research.

TABLE 5.1: AVAILABLE DATASETS

Dataset Name	Class Names	Number of Classes
Dataset-1	bathroom, bathtub, door, en_suite, kitchen, single door, window	7 classes
Dataset-2	space_balconi, space_bedroom, space_dressroom, space_elevator, space_elevator_hall, space_front, space_kitchen,space_multipurpose_space,space_other,space_living_room,space_outdoor_room, space_staircase, space_toilet, background (p.s. all names have prefix as space_)	14 classes
Dataset-3	bath, bathroom storage, chair, coffee table, corner countertop, countertop, dinner table, dish washer, double bed, dryer, kitchen cabine, kitchen sink, refrigerator, shower, sink, sofa, storage, stove, toilet, wardrobe, washing machine	21 classes

5.4 Pre-Processing Module

The Pre-Processing Module incorporates several pre-processing techniques on floor plan data. Firstly, fundamental steps such as Auto-Orient and Resize(640x640) are applied to ensure compatibility and standardization across the dataset. Following these steps, a cleansing process is conducted to enhance the richness of representations within the floor plan object classes. By discarding classes that are not well-represented, the pre-processing stage enhances the variety and informativeness of object classes preserved in all the datasets. Consequently, all the datasets are refined to maintain a minimum number of robustly represented object classes for downstream analysis and model training.

5.5 Detection Module

The Detection Module is a cornerstone within the proposed system design, serving as a pivotal component in the pipeline. This phase explores object detection models, from real-time object detection methodologies predominantly leveraging CNN-based architectures to cutting-edge end-to-end object detection frameworks emphasizing Transformer architectures. The primary objective of this module is to discern and localize various objects within a given floor plan, encompassing elements such as rooms, windows, doors, and more.

The system uses cutting-edge detection models to accurately identify these objects, providing precise bounding box coordinates that encapsulate their spatial extents within the floor plan. Furthermore, the experimentation process within the Detection Module is characterized by a comprehensive evaluation of different model architectures, configurations, and training methodologies. This rigorous exploration enables selecting the most effective and efficient detection model tailored to floor plan data's unique complexities and nuances. The output generated by the Detection Module, consisting of detected objects accompanied by their corresponding bounding box coordinates, serves as input for subsequent stages of analysis and interpretation within the system.

5.6 Post-Processing Module

The Post-Processing Module is introduced following the detection module to post-process the results obtained from the detection module. Upon receiving a list of predictions generated by the detection Module for a given floor plan image, the post-processing module examines these predictions to remove unnecessary fields, ensuring a streamlined and concise output. This step helps enhance the clarity and usability of the detected objects' information. Furthermore, the Post-Processing Module enriches the predicted object list by incorporating vital details such as image dimensions. The module provides valuable context by appending this information to each detected object, facilitating a deeper understanding of the spatial distribution and scale of the identified objects within the floor plan image.

5.7 Segmentation Module

The Segmentation Module plays a pivotal role by leveraging inputs from both the post-processing module and the detected image to generate masks for identified objects. Initially, this module serves as a visual prompt, receiving input images and bounding box coordinates, which are then processed through a mask decoder to generate corresponding masks. By integrating information from the post-processing module, which refines the detected objects and their associated bounding box coordinates, the Segmentation Module ensures accurate alignment between the detected objects and their respective masks. This alignment is crucial for generating precise masks that encapsulate the exact boundaries of the detected objects within the input image.

Through the mask decoder, the segmentation module translates the spatial information the bounding box coordinates provide into masks, effectively delineating the regions corresponding to each detected object. These masks serve as invaluable representations of the spatial extent and segmentation of the identified objects within the input image. In essence, the segmentation module bridges the gap between object detection and segmentation, enabling the extraction of fine-grained spatial information from the detected objects. By generating masks that precisely outline the detected objects, this module facilitates a deeper understanding and analysis of the floor plan imagery, thereby enhancing the system's overall capabilities for automatic floor plan analysis.

5.8 Inference Module

The Inference Module serves as a crucial component in the system architecture, primarily tasked with retrieving the model from the model registry and performing inference tasks on user-provided images. Its primary objective is to ensure the generation of accurate detections and segmentations by utilizing the deployed model. At its core, the inference module is a gateway to the model registry, where trained models are stored and maintained. Upon receiving a request for inference, typically accompanied by a user-provided image, the module retrieves the appropriate model from the registry. This model is carefully selected based on the specific task requirements and the target architecture, ensuring compatibility and optimal performance.

5.9 Web App

The final step of the BuilderFormer is its integration into a user-friendly web application, facilitating seamless interaction and visualization of results for various floor plans, accompanied by comprehensive evaluations. This web app serves as the interface through which users can efficiently access and harness the system's capabilities.

5.10 Summary

The design of the BuilderFormer is a comprehensive solution for automatic floor plan analysis, comprising five key modules that collectively enable accurate detection, segmentation, and analysis of floor plan data. Beginning with the Pre-Processing Module, the system applies essential techniques like Auto-Orient and Resize to standardize the input data, followed by a cleansing process to improve the representation of object classes. The Detection Module explores various object detection models to recognize and localize objects such as rooms, windows, and doors within floor plans. Subsequently, the Post-Processing Module refines detection results by removing unnecessary fields and enriching object lists with essential details. The Segmentation Module is crucial in generating masks for identified objects, bridging the gap between detection and segmentation. The Inference Module handles model retrieval and inference tasks, ensuring accurate detections and segmentations. Finally, the system is integrated into a user-friendly web application, providing users with a seamless interface to interact with the system, visualize results, and conduct evaluations across diverse floor plan datasets. Following the detailed exposition of the BuilderFormer system's architecture and its constituent modules, the subsequent sections will delve into the implementation specifics, evaluation methodologies, and outcomes.

CHAPTER 6

IMPLEMENTATION OF BUILDERFORMER

6.1 Introduction

This section outlines the steps in implementing the BuilderFormer for automated floor plan analysis. It discusses the thorough execution, including data collection and pre-processing methods, the structured modular design of the system, and the careful selection and precise adjustment of pre-trained vision models.

6.2 Data Collection and Preparation

As floor plan research predominantly utilizes proprietary datasets, we procured publicly accessible datasets from the Roboflow universe.

6.2.1 Floor Plan Dataset Format

Floor plan datasets typically comprise images accompanied by corresponding labels, as shown in Figure 6.1 and Figure 6.2. These labels consist of bounding boxes delineated by four parameters:- [x_center, y_center, width, height]. Within this context, x_center and y_center denote the standardized locations of the bounding box's center, while width and height signify the normalized extents of the bounding box.

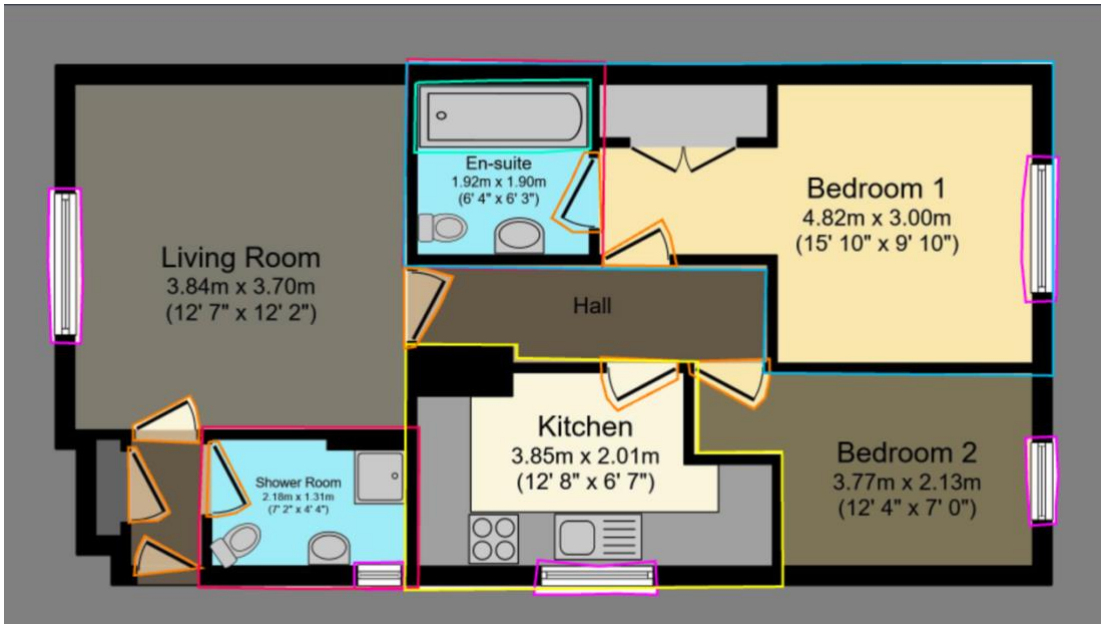


Fig. 6.1: Annotated floor plan image

```
{
"type": "polygon",
"label": "window",
"x": 1666.175,
"y": 358.8935,
"width": 58.46199999999999,
"height": 233.849,
"points": [[1646.688,241.969],[1695.406,243.593],[1693.782,339.406],[1692.158,
474.194],[1643.44,475.818],[1636.944,409.236],[1638.568,313.423]]
}
```


Fig. 6.2: Annotated floor plan image data

6.2.2 Data Preprocessing

Before using raw images, specific pre-processing techniques are applied to ensure they are usable and meaningful, eliminating unwanted distortions and enhancing their quality.

- Auto-Orient:- For several reasons, understanding and adequately handling the EXIF orientation metadata is crucial in floor plan analysis. Firstly, accurate orientation information ensures that the floor plans are displayed correctly to the viewer, maintaining their intended layout and orientation. This is essential for precise analysis and interpretation of the floor plan data. Moreover, consistent handling of orientation metadata allows for uniform processing of floor plan images across different software and platforms. By respecting the EXIF orientation field, we can ensure that the images are consistently displayed and analyzed regardless of the device or software used. Also, proper orientation metadata handling helps prevent errors and inaccuracies in automated analysis tasks, such as object detection or room segmentation, which rely on correctly aligned images for accurate results.

- Resize:- Resizing floor plans for object detection models like YOLO is a crucial preprocessing step with several advantages. Firstly, resizing allows us to standardize the input size of the floor plan images, which is necessary for compatibility with the fixed input size requirements of deep learning models like YOLO. We ensure uniformity in the model's input data by resizing the images to a consistent dimension like 640x640, enabling efficient batch processing and streamlined training. Furthermore, resizing reduces computational complexity and memory usage during the training and inference stages. Smaller-sized images require fewer computational resources, resulting in faster training times and lower memory requirements. This is particularly beneficial for large-scale datasets or resource-constrained environments where computational efficiency is paramount. Additionally, resizing facilitates better generalization of the object detection model. By providing the model with resized images encompassing various scales and viewpoints of the floor plans, we improve its capability to detect objects of different sizes and orientations accurately.

Figures 6.3, 6.4, and 6.5 represent the overview of the datasets we chose for our research project.

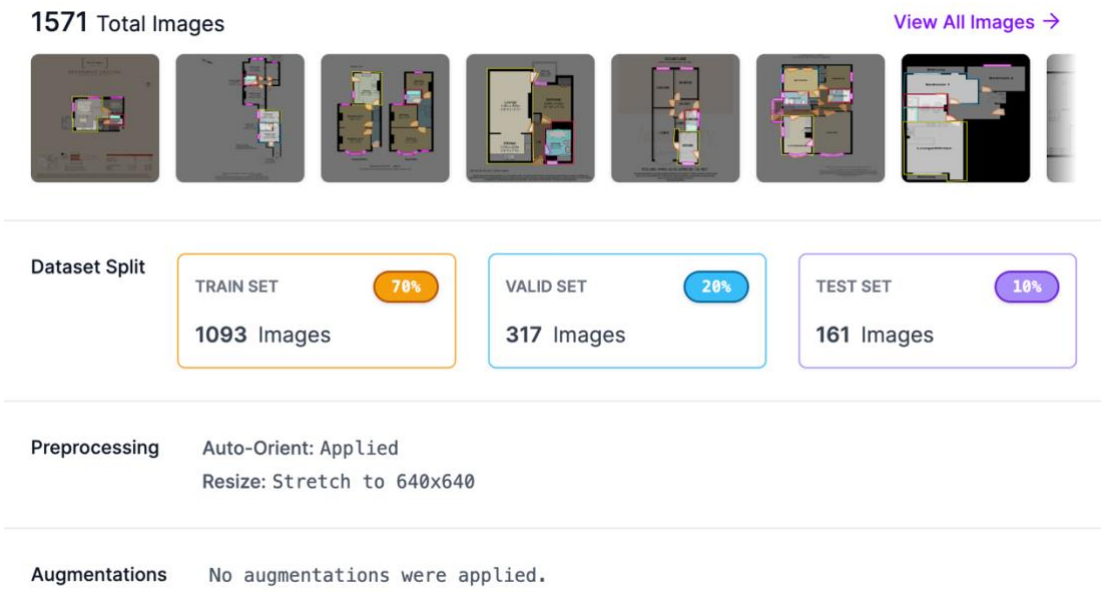


Fig. 6.3: Dataset-1 overview

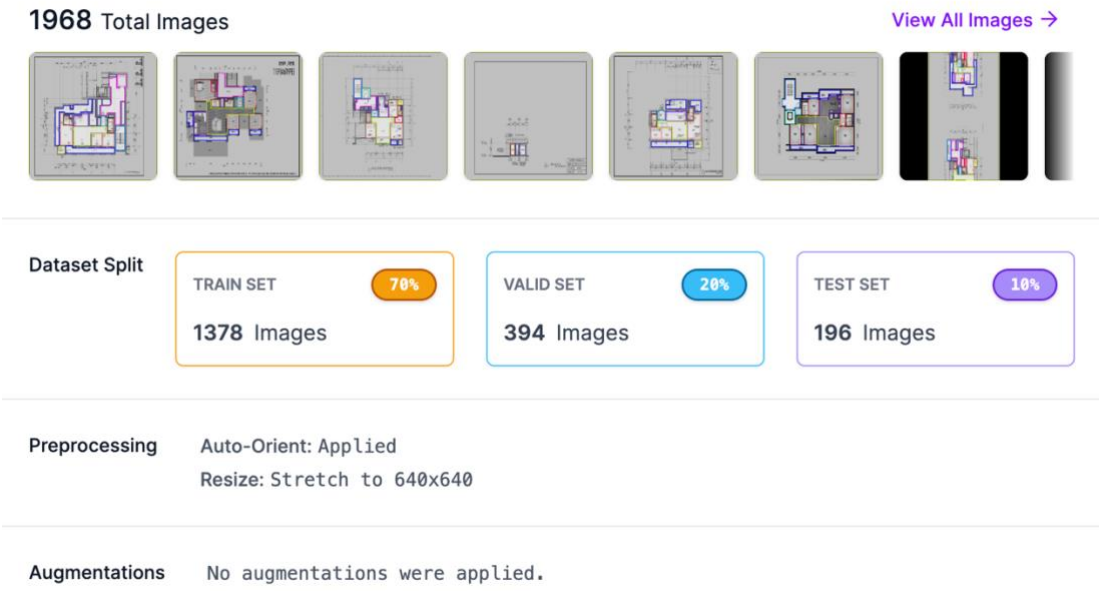


Fig. 6.4: Dataset-2 overview

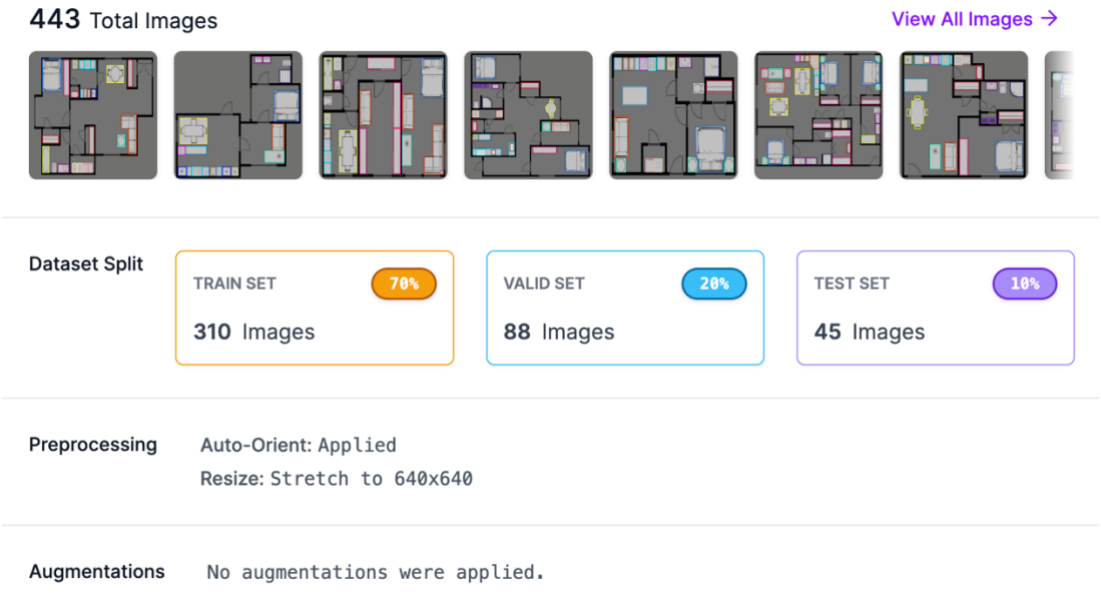


Fig. 6.5: Dataset-3 overview

Data augmentation techniques were not employed when creating the dataset in this research to preserve the holistic integrity of floor plans, as these images require analysis as single, unified entities. Traditional augmentation methods, such as cropping or rotating segments, could disrupt the structural and semantic relationships within floor plans, potentially compromising the accuracy of results. Therefore, we worked exclusively with the original, unaltered images to ensure accurate interpretation.

6.3 Detection Module Implementation

An array of object detection models underwent experimentation within the detection module, including YOLOv8 and RT-DETR. Visual representations of each model's summary are depicted in Figures 6.6 and 6.7, encapsulating their respective architectures.

```
                   from  n    params  module                                       arguments
  0                  -1  1      1856  ultralytics.nn.modules.conv.Conv             [3, 64, 3, 2]
  1                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  2                  -1  3    279808  ultralytics.nn.modules.block.C2f             [128, 128, 3, True]
  3                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  4                  -1  6   2101248  ultralytics.nn.modules.block.C2f             [256, 256, 6, True]
  5                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  6                  -1  6   8396800  ultralytics.nn.modules.block.C2f             [512, 512, 6, True]
  7                  -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
  8                  -1  3   4461568  ultralytics.nn.modules.block.C2f             [512, 512, 3, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  3   4723712  ultralytics.nn.modules.block.C2f             [1024, 512, 3]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  3   1247744  ultralytics.nn.modules.block.C2f             [768, 256, 3]
 16                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  3   4592640  ultralytics.nn.modules.block.C2f             [768, 512, 3]
 19                  -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  3   4723712  ultralytics.nn.modules.block.C2f             [1024, 512, 3]
 22        [15, 18, 21]  1   5593594  ultralytics.nn.modules.head.Detect           [14, [256, 512, 512]]
Model summary: 365 layers, 43640634 parameters, 43640618 gradients, 165.5 GFLOPs
```

Fig. 6.6: YOLOv8 model summary

```
                   from  n    params  module                                       arguments
  0                  -1  1     25248  ultralytics.nn.modules.block.HGStem          [3, 32, 48]
  1                  -1  6    155072  ultralytics.nn.modules.block.HGBlock         [48, 48, 128, 3, 6]
  2                  -1  1      1408  ultralytics.nn.modules.conv.DWConv           [128, 128, 3, 2, 1, False]
  3                  -1  6    839296  ultralytics.nn.modules.block.HGBlock         [128, 96, 512, 3, 6]
  4                  -1  1      5632  ultralytics.nn.modules.conv.DWConv           [512, 512, 3, 2, 1, False]
  5                  -1  6   1695360  ultralytics.nn.modules.block.HGBlock         [512, 192, 1024, 5, 6, True, False]
  6                  -1  6   2055808  ultralytics.nn.modules.block.HGBlock         [1024, 192, 1024, 5, 6, True, True]
  7                  -1  6   2055808  ultralytics.nn.modules.block.HGBlock         [1024, 192, 1024, 5, 6, True, True]
  8                  -1  1     11264  ultralytics.nn.modules.conv.DWConv           [1024, 1024, 3, 2, 1, False]
  9                  -1  6   6708480  ultralytics.nn.modules.block.HGBlock         [1024, 384, 2048, 5, 6, True, False]
 10                  -1  1    524800  ultralytics.nn.modules.conv.Conv             [2048, 256, 1, 1, None, 1, 1, False]
 11                  -1  1    789760  ultralytics.nn.modules.transformer.AIFI      [256, 1024, 8]
 12                  -1  1     66048  ultralytics.nn.modules.conv.Conv             [256, 256, 1, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14                   7  1    262656  ultralytics.nn.modules.conv.Conv             [1024, 256, 1, 1, None, 1, 1, False]
 15            [-2, -1]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 16                  -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 17                  -1  1     66048  ultralytics.nn.modules.conv.Conv             [256, 256, 1, 1]
 18                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 19                   3  1    131584  ultralytics.nn.modules.conv.Conv             [512, 256, 1, 1, None, 1, 1, False]
 20            [-2, -1]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 22                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 23            [-1, 17]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 24                  -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 25                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 26            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 27                  -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 28        [21, 24, 27]  1   7330622  ultralytics.nn.modules.head.RTDETRDecoder    [14, [256, 256, 256]]
rt-detr-l summary: 673 layers, 32834846 parameters, 32834846 gradients
```

Fig. 6.7: RT-DETR model summary

Subsequently, we leveraged the Ultralytics library to load the models and initiate training, benefiting from its user-friendly interface tailored for model training. The Ultralytics library stands out for its comprehensive functionality, offering seamless integration with popular object detection architectures such as YOLOv8 and RT-DETR. This library simplifies the training process by providing extensive tools for data handling, model configuration, hyperparameter tuning, and performance evaluation. Its intuitive interface facilitates swift experimentation and iteration, enabling researchers to focus on refining model architectures and optimizing training strategies without being bogged down by technical intricacies.

Figures 6.8 and 6.9 below show the code to train both models separately.

Fig. 6.8: YOLOv8 model training

Fig. 6.9: RT-DETR model training

We trained the models using the NVIDIA Tesla T4 GPU with the below properties.

- Train/Valid/Test Ratio:- 70:20:10
- Epochs:- 100
- Batch Size:- 16
- Albumentations:- Applying transformations such as,
    - Blur (with a likelihood of 0.01 and blur boundaries between 3 and 7)
    - CLAHE (with a likelihood of 0.01, clip boundaries ranging from 1 to 4.0, and tile grid size of 8x8)
    - MedianBlur (with a likelihood of 0.01 and blur boundaries between 3 and 7)
    - ToGray (with a likelihood of 0.01)

- Optimizers:- AdamW optimizer with a learning rate 0.000556 and momentum of 0.9.
    - Group 1:- 97 weights with no weight decay
    - Group 2:- 104 weights with weight decay of 0.0005
    - Group 3:- 103 biases with no weight decay

Let's examine the properties mentioned above in detail using the following points.

- Epochs:- An epoch represents a single iteration of the dataset passing via the model during training. Specifying 100 epochs means the model will iterate over the entire dataset 100 times during training. In floor plan analysis, training for multiple epochs enables the model to progressively learn and refine its comprehension of floor plan features, resulting in enhanced performance over time.
- Batch Size:- The batch size dictates the number of samples the model processes in each training iteration. A batch size of 16 means the model processes 16-floor plan images during training. Choosing an appropriate batch size balances computational efficiency and model stability. A batch size 16 in floor plan analysis allows the model to efficiently learn from a subset of the dataset while benefiting from GPU parallelism.
- Albumentations:- Albumentations [38] is a popular library for image augmentation, offering a wide range of transformations to enhance the diversity of the training data. In floor plan analysis, augmentations such as blur, median blur, grayscale conversion, and contrast-limited adaptive histogram equalization (CLAHE) can simulate variations in floor plan images due to different lighting conditions, image quality, and perspective distortions. This helps the model generalize better to unseen variations in real-world floor plan data.
- Optimizer:- The optimizer adjusts the model's parameters based on the computed gradients during training. AdamW (Adam with Weight Decay) is a variant of the Adam optimizer that includes weight decay regularization to prevent overfitting. The specified learning rate (lr=0.000556) and momentum (0.9) control the magnitude and direction of parameter updates. The choice of optimizer and its hyperparameters, tuned for floor plan analysis, influences the model's convergence speed and generalization performance.

The output of the detection module is depicted in Figure 6.10.

Fig. 6.10: Detection module predictions

The evaluation part of these detection models will be discussed in the Evaluation chapter.

6.4 Post-Processing Module Implementation

After receiving a list of predictions from the Detection Module for a floor plan image, the Post-Processing Module scrutinizes these predictions to eliminate redundant information, resulting in a more focused and concise output. This process aims to improve the clarity and usability of the detected objects' information. Moreover, the Post-Processing Module enriches the predicted object list by including essential details like image dimensions. Incorporating this information into each detected object provides valuable context, facilitating a deeper understanding of the spatial distribution and size of the identified objects within the floor plan image.

The following functions in Figures 6.11 and 6.12 show how we remove unnecessary fields from the prediction result JSON.

```python
# Remove unnecessary fields from results
def remove_unnecessary_fields(predictions):
    """
    Removes unnecessary fields ("tracker_id" and "class_confidence") from a list of predictions.

    Args:
        predictions: A list of dictionaries containing object predictions.

    Returns:
        A new list of dictionaries with the unnecessary fields removed.
    """

    cleaned_predictions = []
    for prediction in predictions:
        cleaned_prediction = {key: value for key, value in prediction.items() if key not in ("tracker_id", "class_confidence")}
        cleaned_predictions.append(cleaned_prediction)
    return {'predictions': cleaned_predictions}
```

Fig. 6.11: Remove unnecessary fields function

```python
# Update Image Dimensions
def update_image_dimensions(predictions):
    """
    Adds an "image" dictionary with "width" and "height" fields to each prediction.

    Args:
        predictions: A list of dictionaries containing object predictions.

    Returns:
        The modified list of predictions with the added "image" field and dimensions.
    """

    predictions["image"] = {"width": 640, "height": 640}
    return predictions

updated_predictions = update_image_dimensions(cleaned_predictions)
print(updated_predictions)
```

Fig. 6.12: Update image dimensions function

After that, we plot the detected floor plan objects using the Roboflow Supervision Library. Supervision provides a simplified solution for annotating predictions generated by various object detection and segmentation models. Ultimately, we can annotate the image using the predictions. When dealing with an object detection model, we utilize the sv.BoundingBoxAnnotator and sv.LabelAnnotator classes. Alternatively, if employing a segmentation model, sv.MaskAnnotator serves as a suitable alternative to sv.BoundingBoxAnnotator enables the drawing of masks instead of boxes. Figures 6.13 and 6.14 represent the function we used to plot the predicted floor plan objects and the plotted image.

```
# Plot Detected Floor Plan Objects using Roboflow Supervision Library
labels = [item["class"] for item in result["predictions"]]

detections = sv.Detections.from_roboflow(updated_predictions)

label_annotator = sv.LabelAnnotator()
box_annotator = sv.BoxAnnotator()
mask_annotator = sv.MaskAnnotator()

image = cv2.imread("/content/C4_png_jpg.rf.63a3de24ed359c7f18079aa84bc2653a.jpg")

detected_image = box_annotator.annotate(
    scene=image, detections=detections)
detected_image = label_annotator.annotate(
    scene=detected_image, detections=detections, labels=labels)

sv.plot_image(image=detected_image, size=(16, 16))
```

Fig. 6.13: Roboflow supervision detection plot function

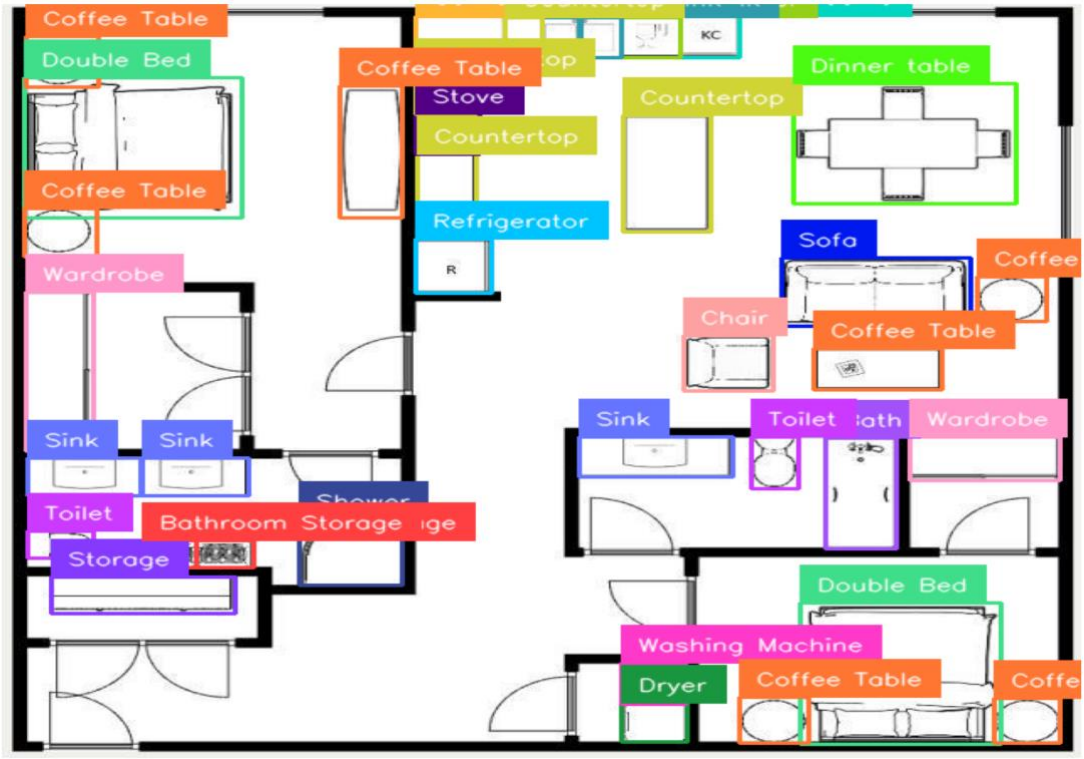


Fig. 6.14: Plotted detection objects

6.5 Segmentation Module Implementation

We adopted the Segment Anything Model in the Segmentation Module to create segmentation masks to predict objects from the object detection module. We utilized the SAM VIT-H Model for the segmentation module.

First, as in Figure 6.15, we download the SAM weights and load the model to our CUDA-enabled environment.

```
# Download SAM weights
!mkdir -p {HOME}/weights
!wget -q https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth -P {HOME}/weights
```

```
# A quick sanity check to see if the weights are downloaded
CHECKPOINT_PATH = os.path.join(HOME, "weights", "sam_vit_h_4b8939.pth")
print(CHECKPOINT_PATH, "; exist:", os.path.isfile(CHECKPOINT_PATH))
```

```
/content/weights/sam_vit_h_4b8939.pth ; exist: True
```

```
# Loading the model
DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
MODEL_TYPE = "vit_h"
sam = sam_model_registry[MODEL_TYPE](checkpoint=CHECKPOINT_PATH).to(device=DEVICE)
mask_predictor = SamPredictor(sam)
```

Fig. 6.15: Loading sam model

Then, as in Figure 6.16, we create a function to encode the image alongside bounding box predictions as the prompt for the Segment Anything model.

```
# SAM Segment Function Creation
import numpy as np
from segment_anything import SamPredictor


def segment(sam_predictor: SamPredictor, image: np.ndarray, xyxy: np.ndarray) -> np.ndarray:
    sam_predictor.set_image(image)
    result_masks = []
    for box in xyxy:
        masks, scores, logits = sam_predictor.predict(
            box=box,
            multimask_output=True
        )
        index = np.argmax(scores)
        result_masks.append(masks[index])
    return np.array(result_masks)
```

```
mask_predictor = SamPredictor(sam)
```

Fig. 6.16: Prompt encode function

After that, we plotted the segmented mask for each object using the Roboflow Supervision Library, as in Figure 6.17.

```python
# Plot Segmented Floor Plan Objects using Roboflow Supervision Library
import cv2

# convert detections to masks
detections.mask = segment(
    sam_predictor=mask_predictor,
    image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB),
    xyxy=detections.xyxy
)

# annotate image with detections
box_annotator = sv.BoxAnnotator()
mask_annotator = sv.MaskAnnotator()
labels = labels
annotated_image = mask_annotator.annotate(scene=image.copy(), detections=detections)
annotated_image = box_annotator.annotate(scene=annotated_image, detections=detections, labels=labels)

%matplotlib inline
sv.plot_image(annotated_image, (16, 16))
```

Fig. 6.17: Roboflow supervision segmentation plot function

Figure 6.18 represents the plotted segmentation masks in the image.

Fig. 6.18: Plotted segmentation masks

6.6 Model Registry

We utilized Roboflow Deploy to deploy our trained models and keep it as the Model Registry to store all our trained models. Roboflow Deploy is a platform crafted to facilitate the deployment of trained computer vision models across different environments. It provides a range of deployment choices tailored to our requirements, from basic testing on our local system to orchestrating models for large-scale usage in production settings. Figures 6.19 and 6.20 show the function of deploying the model to Roboflow and how the deployed model is visualized in the Roboflow.

```
project.version(dataset.version).deploy(model_type="yolov8", model_path=f"{HOME}/runs/detect/train6/")
```

Fig. 6.19: Roboflow deploy function

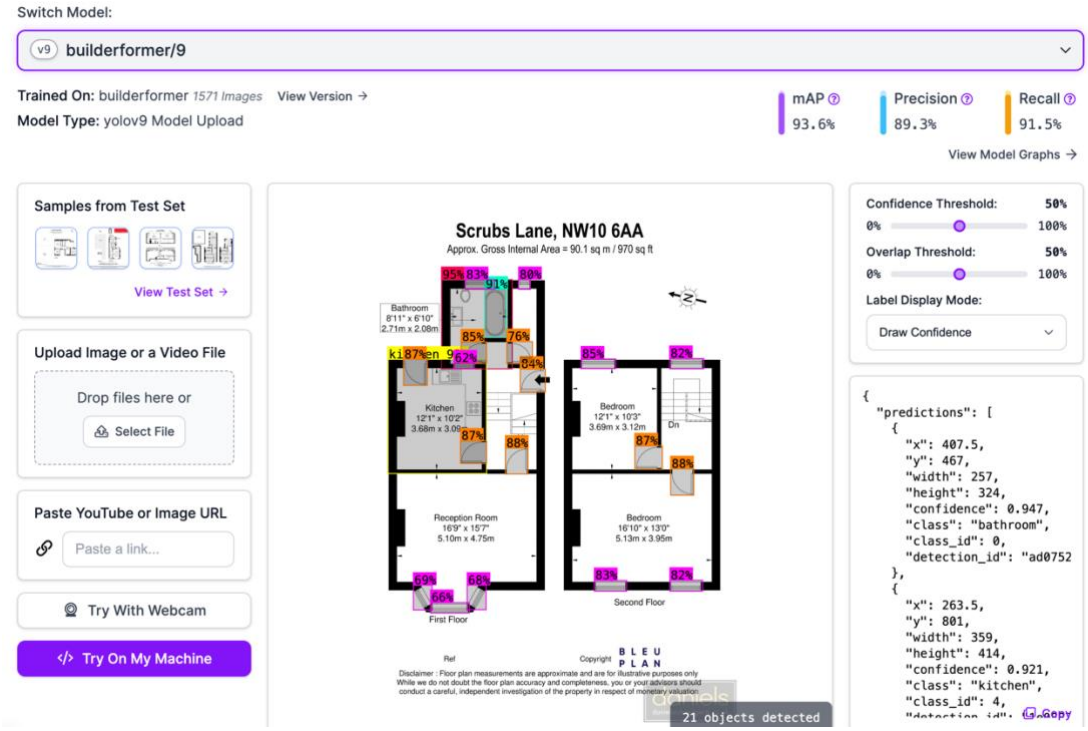


Fig. 6.20: Roboflow visualize models dashboard

6.7 Web App Implementation

Figures 6.21, 6.22 and 6.23 represent the screenshots of the implemented web application. The developed web app's code is attached to Appendix A.

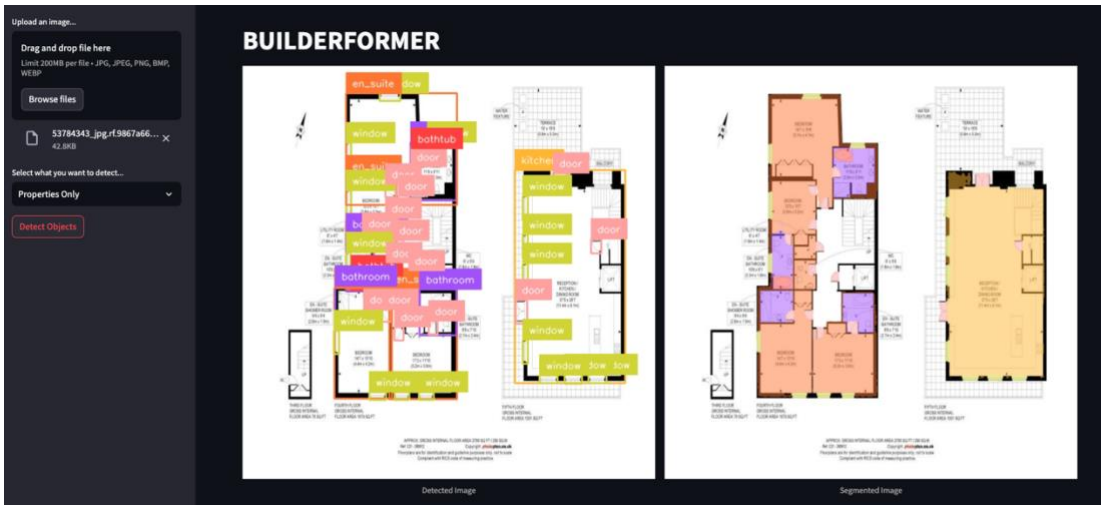


Fig. 6.21: BuilderFormer-1

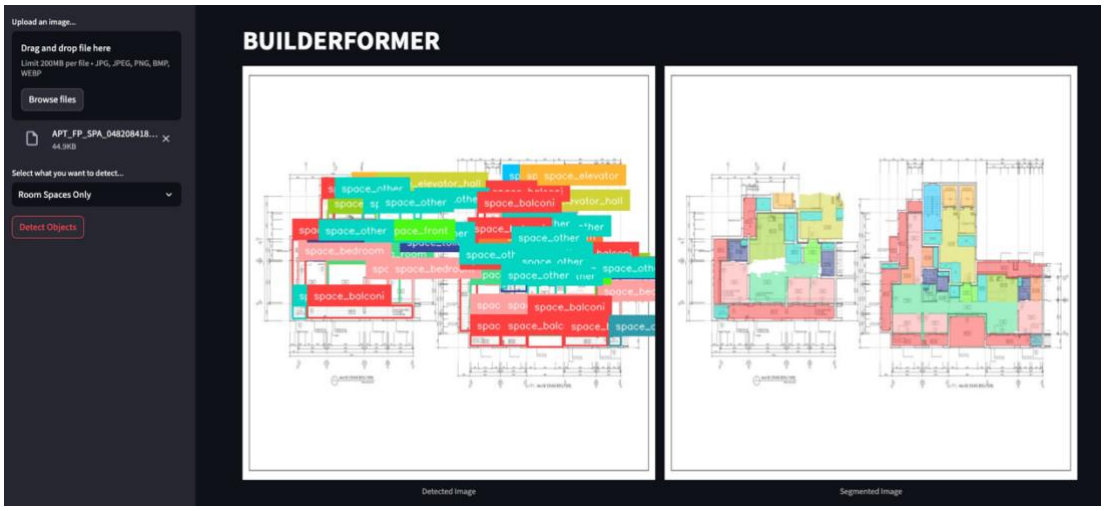


Fig. 6.22: BuilderFormer-2

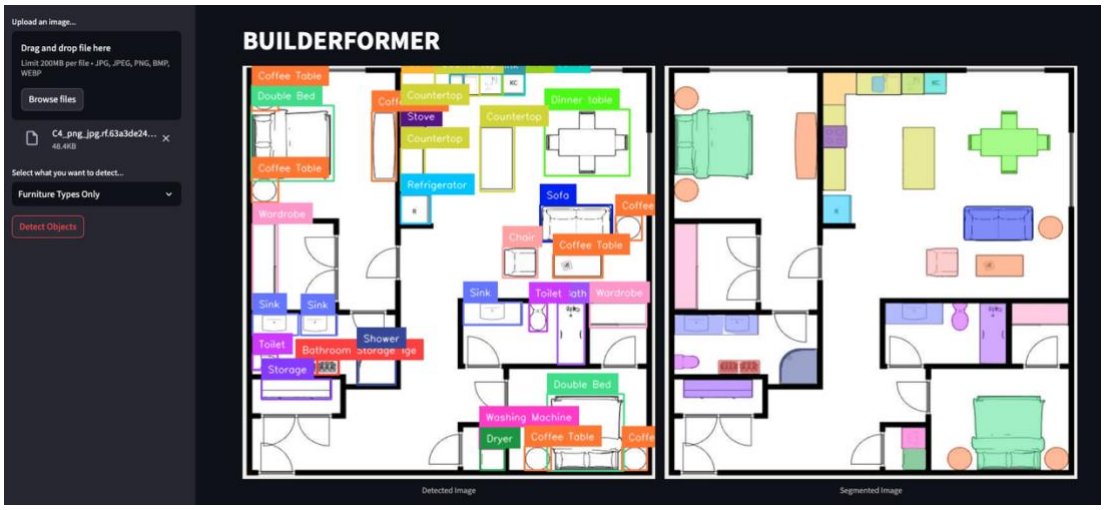


Fig. 6.23: BuilderFormer-3

6.8 Summary

Within this chapter, we outlined the implementation process of the BuilderFormer, explicitly designed for automatic floor plan analysis. The implementation comprised several vital steps, including data collection and preprocessing, development of the object detection module, post-processing module, development of the segmentation module, and model deployment. The data collection and preprocessing phases were meticulously executed to ensure the quality and relevance of the floor plan datasets. Within the object detection module, we explored the real-time performance of YOLOv8 and RT-DETR models to generate bounding box object detections for the segmentation module. The segmentation module is equipped with SAM to produce masks corresponding to the bounding box predictions. Finally, the inference module employed fine-tuned models to generate accurate floor plan object predictions. The subsequent chapter will evaluate the system's performance, accuracy, and efficiency, along with an in-depth analysis of the obtained results.

CHAPTER 7

EVALUATION OF BUILDERFORMER

7.1 Introduction

The primary objective of this evaluation chapter is to thoroughly assess the performance, accuracy, and efficiency of the developed BuilderFormer tailored for the automatic floor plan analysis. Within this chapter, we introduce the evaluation methodology employed, detail the metrics utilized for assessment, and outline the overarching goals of the evaluation process. By examining the evaluation results, we aim to gain valuable insights into the system's efficacy in delivering precise results and its efficiency in response time. By meticulously evaluating the system's performance, we gain a holistic understanding of its capabilities, strengths, and limitations. This enables us to draw informed conclusions regarding its effectiveness in fulfilling the automatic floor plan analysis.

7.2 Experimental Datasets

As discussed in the preceding chapters, we employed three distinct datasets to substantiate the robustness of our proposed approach for automated floor plan analysis. With each dataset, we evaluated the results using proper evaluation metrics.

7.3 Evaluation Metrics

We utilized Mean Average Precision, Precision and Recall as the primary evaluation metrics for the object detection module.

- Mean Average Precision (mAP): Mean Average Precision is a comprehensive metric that evaluates the performance of object detection models across different classes and various levels of confidence thresholds. It calculates the average precision for each class and then averages them, providing an overall measure of the model's accuracy in detecting objects. A higher mAP indicates better performance.
- Precision:- Precision quantifies the ratio of accurately detected objects among all objects predicted by the model. It is computed as the proportion of true positives to the sum of true and false positives. Precision reflects the model's

ability to avoid false positives, i.e., incorrectly identifying non-existent objects as positives. A higher precision indicates fewer false positives and better accuracy in object detection.

- Recall:- Recall, also called sensitivity or actual positive rate, measures the model's capability to identify all relevant objects within the dataset. It is calculated as the ratio of true positives to the sum of true positives and false negatives. Recall measures the completeness of object detection, indicating how well the model captures all instances of a particular class. A higher recall suggests better coverage and completeness in object detection.

7.4 Experimental Results

Following Table 7.1 represents our experimental results on three distinct datasets.

TABLE 7.1: EXPERIMENTAL RESULTS

Dataset with no. of classes	Configuration		Evaluation Metrics			Inference Time Per Image
	Detection	Segmentation	mAP	Precision	Recall	
Dataset-1 (7 classes)	YOLO-v8 (Large)	SAM (Vit-H)	93.6%	89.3%	91.5%	Speed: 0.4ms preprocess, 17.8ms inference, 0.0ms loss, 4.0ms postprocess per image
	RT-DETR (Large)	SAM (Vit-H)	93.6%	89%	91%	Speed: 0.4ms preprocess, 22.1ms inference, 0.0ms loss, 7.1ms postprocess per image

Dataset-2 (14 classes)	YOLO-v8 (Large)	SAM (Vit-H)	93.2%	93.4%	89.4%	Speed: 0.4ms preprocess, 17.1ms inference, 0.0ms loss, 8.2ms postprocess per image
	RT-DETR (Large)	SAM (Vit-H)	92.9%	93.1%	86.8%	Speed: 0.2ms preprocess, 15.2ms inference, 0.0ms loss, 0.3ms postprocess per image
Dataset-3 (21 classes)	YOLO-v8 (Large)	SAM (Vit-H)	96.2%	95.5%	95.6%	Speed: 9.2ms preprocess, 41.7ms inference, 0.0ms loss, 12.2ms postprocess per image
	RT-DETR (Large)	SAM (Vit-H)	96.2%	94.7%	95.6%	Speed: 0.2ms preprocess, 17.7ms inference, 0.0ms loss, 1.1ms postprocess per image

7.4.1 Results From Experiment 1

In Experiment 1, conducted on Dataset-1, the object detection model YOLOv8 Large and segmentation model SAM VIT-H were employed, yielding results with 93.6% of mAP, 89.3% of Precision, and 91.5% of Recall. The confusion matrix elucidates the model's proficiency in object recognition. The confusion matrix generated on the results of Dataset-1 shows that all 7 classes identified with a better accuracy rate. Furthermore, we created visualizations of evaluation metrics graphs to illustrate model results, including box_loss, cls_loss, and dfl_loss. All the generated visualization graphs exhibit smooth curves, indicating that the trained model performs well.

Figure 7.1 depicts the experiment 1 confusion matrix. In Figure 7.2, we present the evaluation metrics graph for experiment 1.

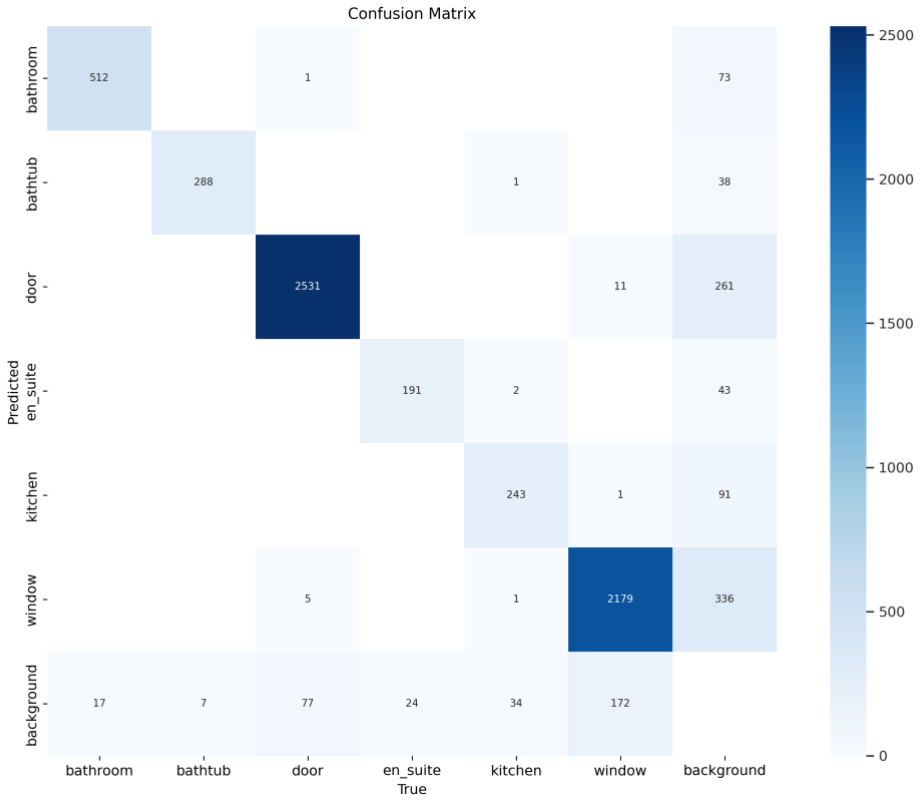


Fig. 7.1: Experiment 1 confusion matrix

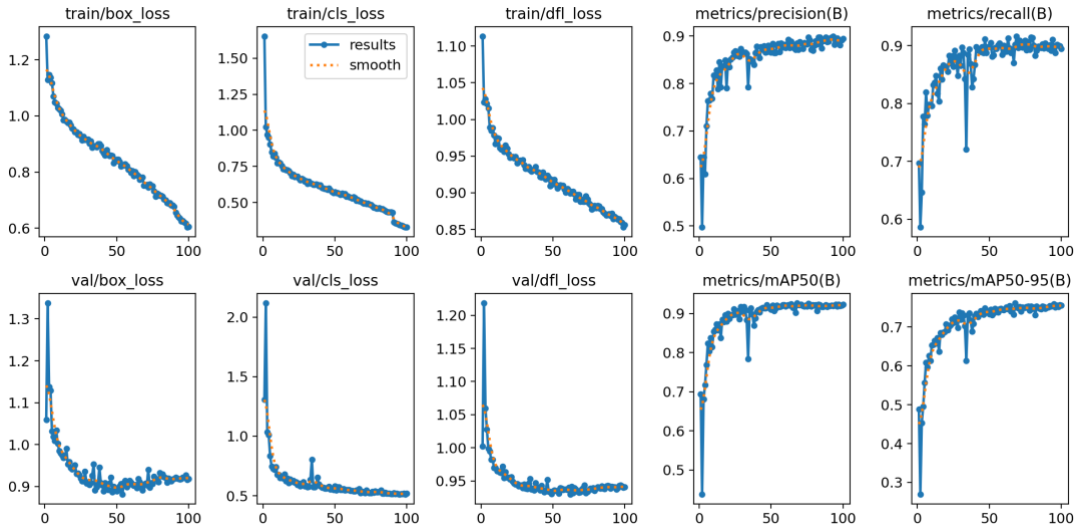


Fig. 7.2: Experiment 1 evaluation metrics graphs

Figure 7.3 illustrates the visualization of testing set results on experiment 1, which is about Dataset-1.

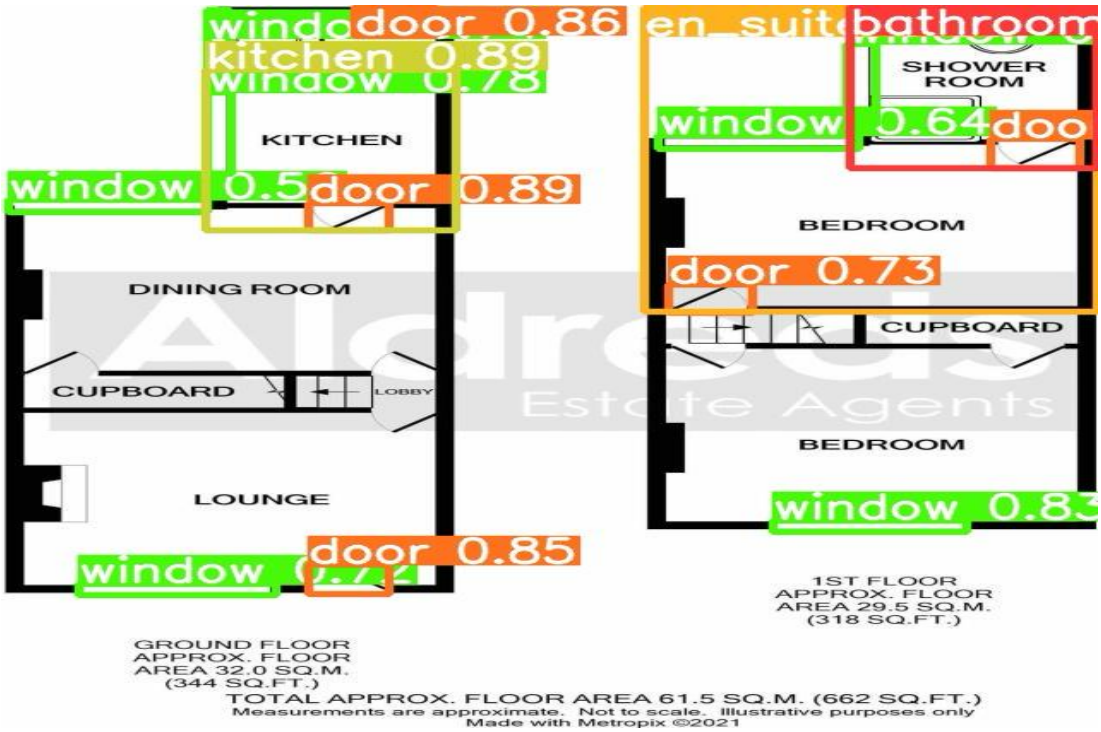


Fig. 7.3: Experiment 1 testing set visualization

7.4.2 Results From Experiment 2

Experiment 2, conducted on Dataset-2, utilized the YOLOv8 Large object detection model and SAM VIT-H segmentation model, resulting in 93.2% of mAP, 93.4% of Precision, and 89.4% of Recall. The confusion matrix provides insights into the model's object detection capabilities, showing improved accuracy rates across all 14 identified classes. Additionally, visualization graphs of evaluation metrics such as box_loss, cls_loss, and dfl_loss were generated to depict model performance. These graphs exhibit smooth curves, indicating the satisfactory performance of the trained model.

Following Figure 7.4 depicts the experiment 2 confusion matrix. In Figure 7.5, we present the evaluation metrics graph for experiment 2.

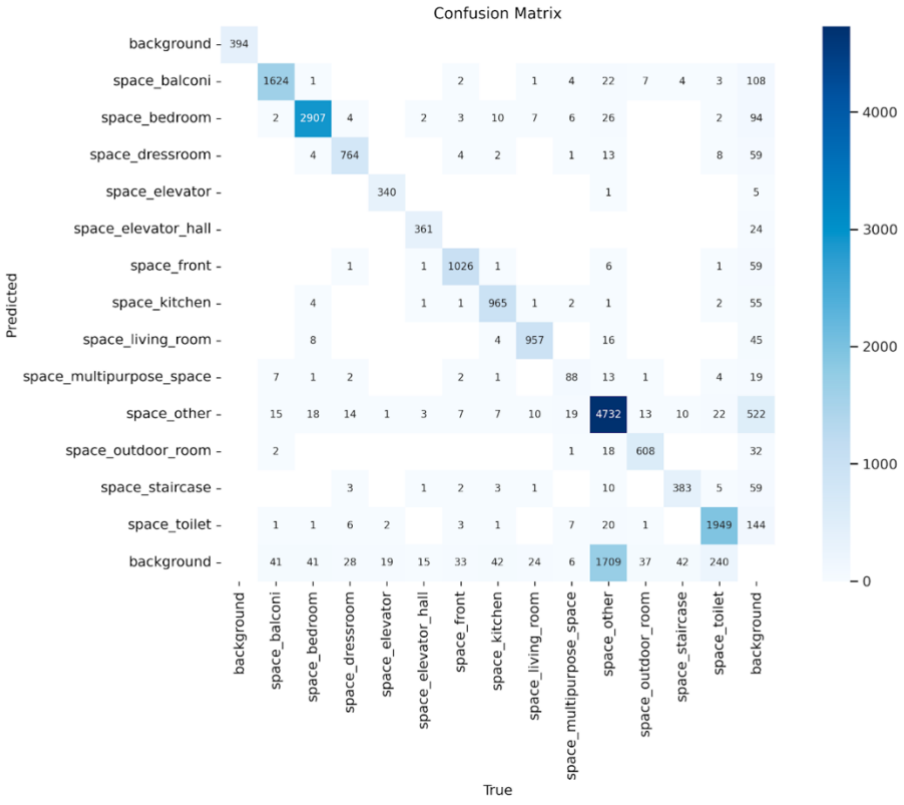


Fig. 7.4: Experiment 2 confusion matrix

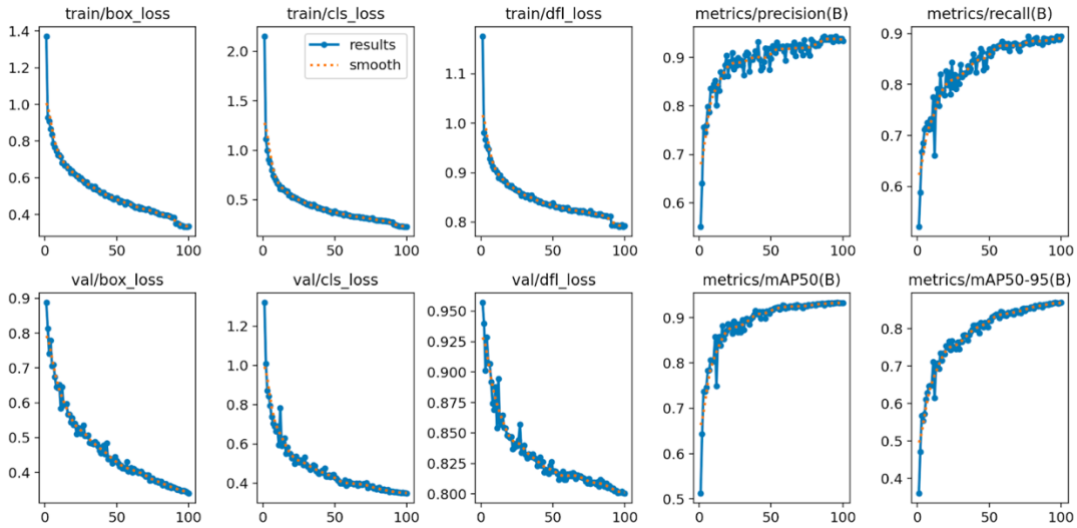


Fig. 7.5: Experiment 2 evaluation metrics graphs

Following Figure 7.6 illustrates the visualization of testing set results on Dataset-2.

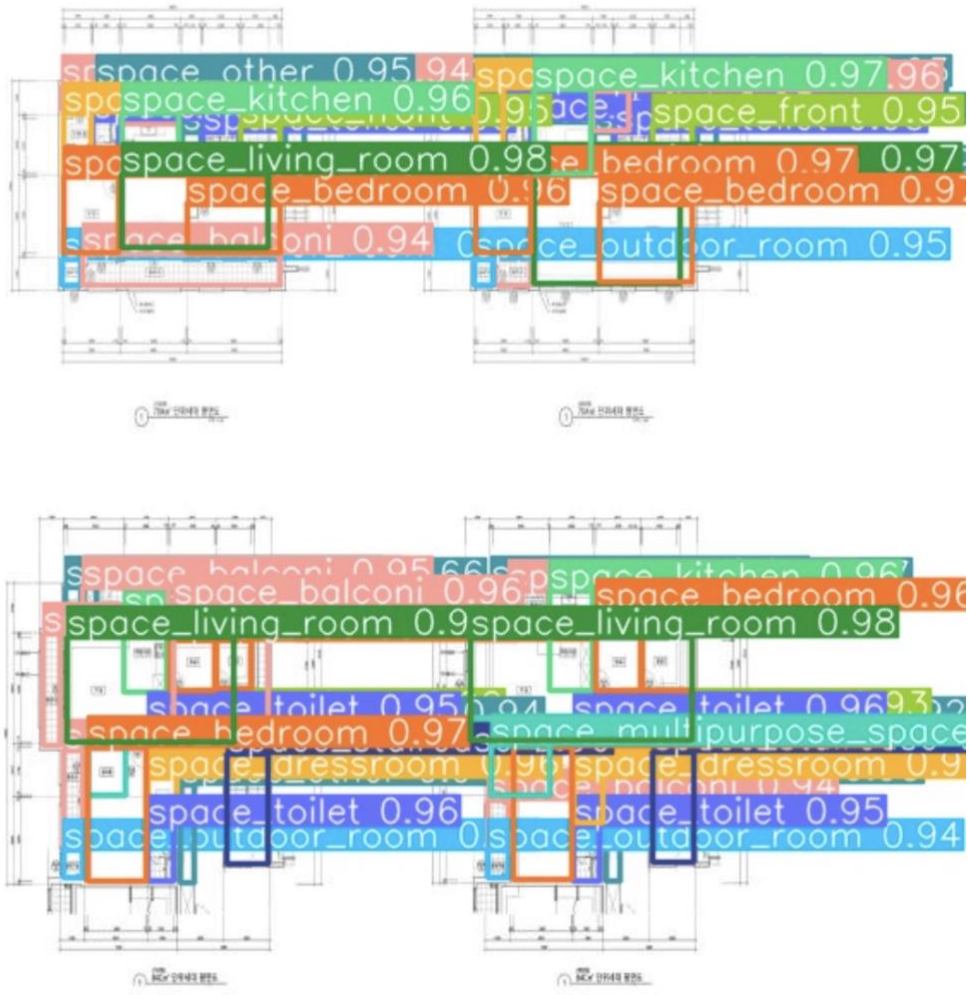


Fig. 7.6: Experiment 2 testing set visualization

7.4.3 Results From Experiment 3

Experiment 3, carried out on Dataset-3, utilized the YOLOv8 Large object detection model and SAM VIT-H segmentation model, resulting in 96.2% of mAP, 95.5% of Precision, and 96.5% of Recall. The confusion matrix offers insights into the model's object detection capabilities, demonstrating enhanced accuracy rates across all 21 identified classes. Moreover, visualization graphs of evaluation metrics like box_loss, cls_loss, and dfl_loss were generated to illustrate model performance. These graphs exhibit smooth curves, indicating the satisfactory performance of the trained model.

Following Figure 7.7 depicts the experiment 3 confusion matrix. In Figure 7.8, we present the evaluation metrics graph for experiment 3.

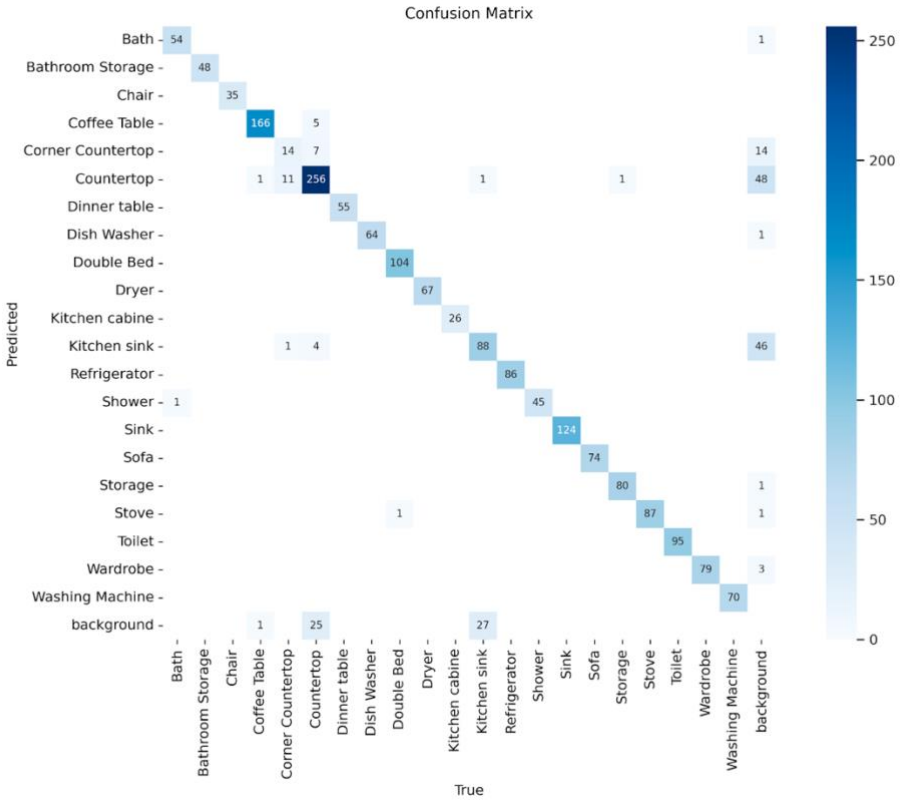


Fig. 7.7: Experiment 3 confusion matrix

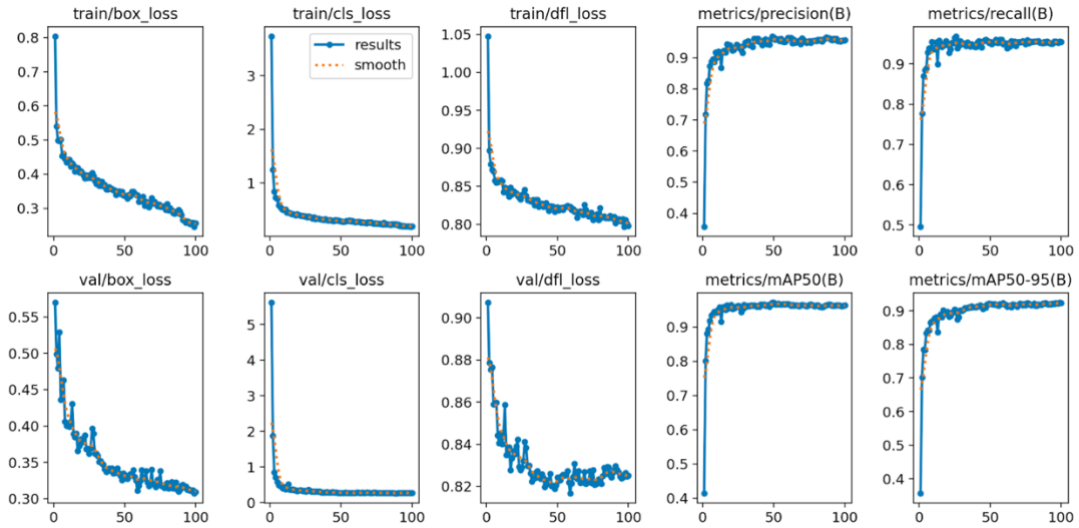


Fig. 7.8: Experiment 3 evaluation metrics graphs

Following Figure 7.9 illustrates the visualization of testing set results on Dataset-3.

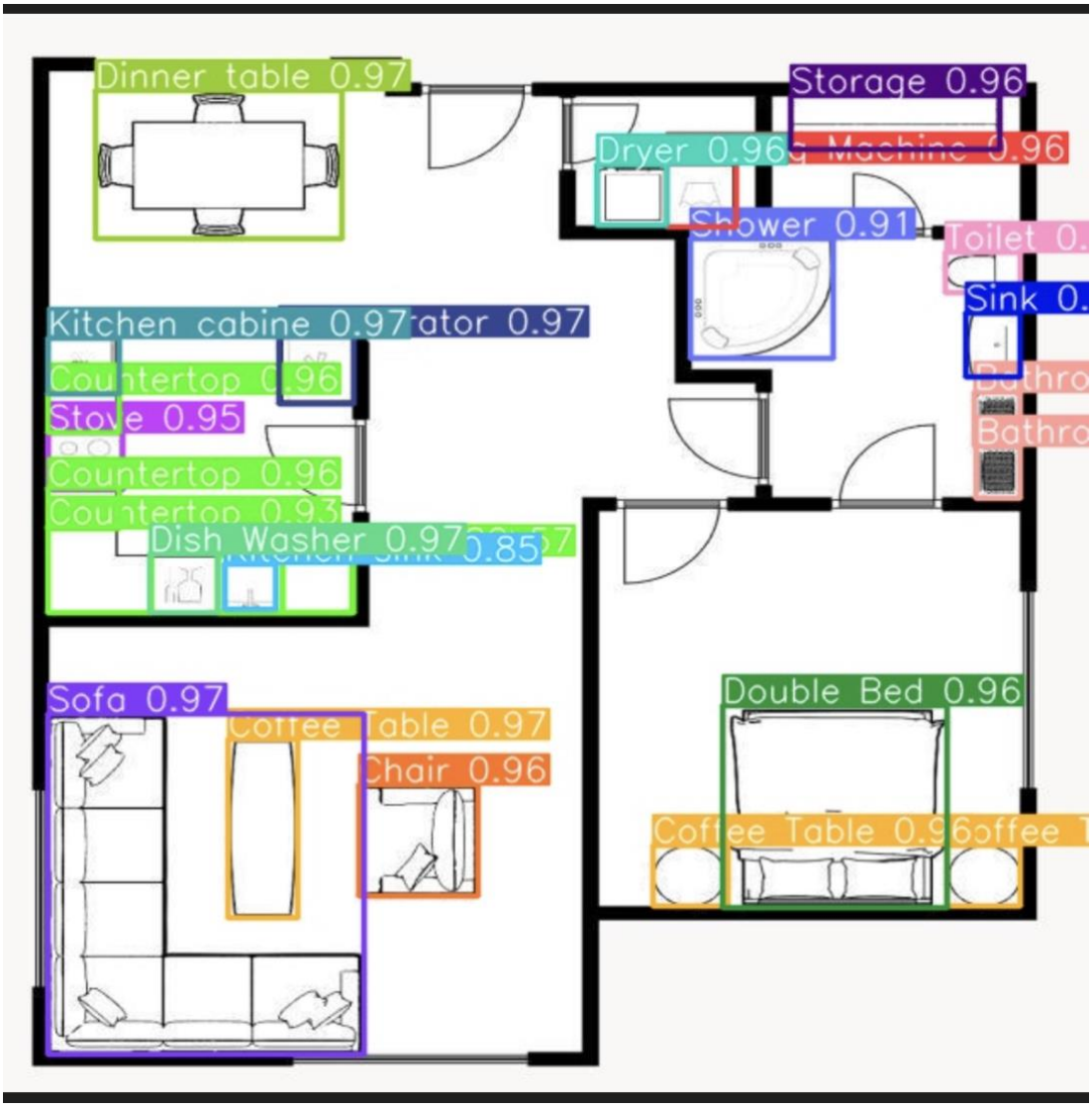


Fig. 7.9: Experiment 3 testing set visualization

7.4.4 Results From Experiment 4

In Experiment 4, conducted on Dataset-1, the object detection model RT-DETR Large and segmentation model SAM VIT-H were employed, yielding results with 93.6% of mAP, 89% of Precision, and 91% of Recall. The confusion matrix elucidates the model's proficiency in object recognition. The confusion matrix generated on the results of Dataset-1 shows that all 7 classes identified with a better accuracy rate. Furthermore, we created visualizations of evaluation metrics graphs to illustrate model results, including box_loss, cls_loss, and dfl_loss. All the generated visualization graphs exhibit smooth curves, indicating that the trained model performs well.

Figure 7.10 depicts the experiment 4 confusion matrix. In Figure 7.11, we present the evaluation metrics graph for experiment 4.

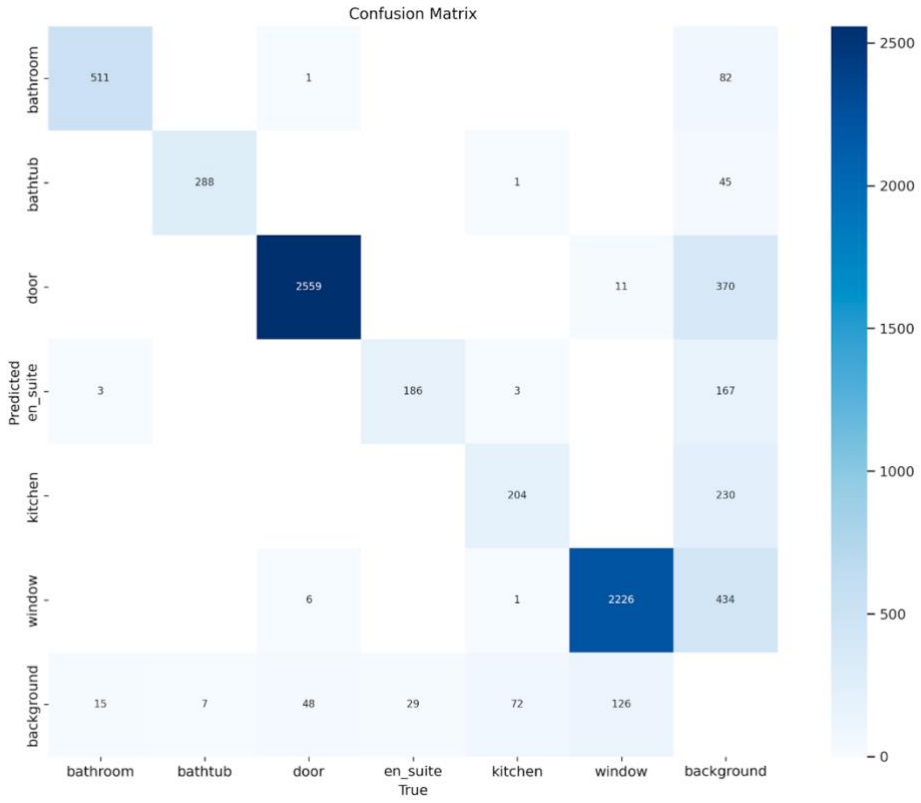


Fig. 7.10: Experiment 4 confusion matrix

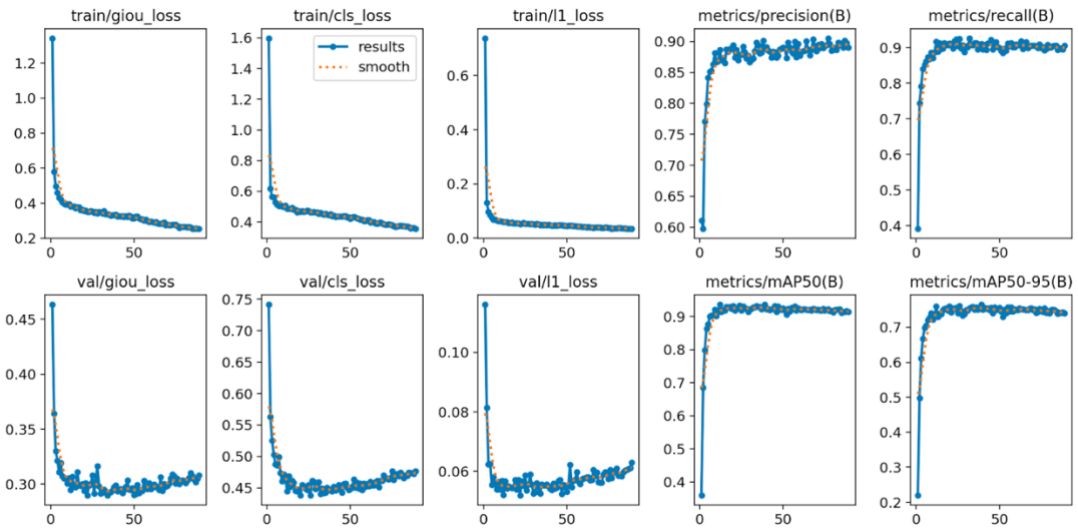


Fig. 7.11: Experiment 4 evaluation metrics graphs

Figure 7.12 illustrates the visualization of testing set results on experiment 4, which is about Dataset-1.

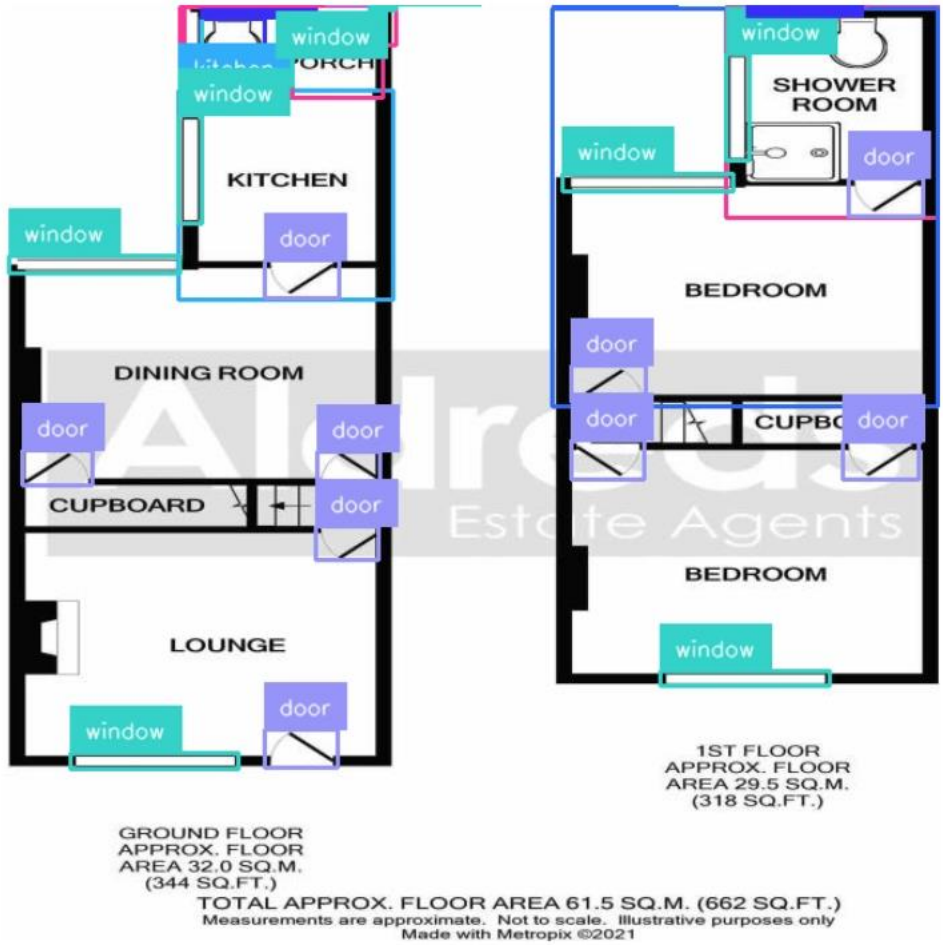


Fig. 7.12: Experiment 4 testing set visualization

7.4.5 Results From Experiment 5

Experiment 5, conducted on Dataset-2, utilized the RT-DETR Large object detection model and SAM VIT-H segmentation model, resulting in 92.9% of mAP, 93.1% of Precision, and 86.8% of Recall. The confusion matrix provides insights into the model's object detection capabilities, showing improved accuracy rates across all 14 identified classes. Additionally, visualization graphs of evaluation metrics such as box_loss, cls_loss, and dfl_loss were generated to depict model performance. These graphs exhibit smooth curves, indicating the satisfactory performance of the trained model. Figure 7.13 and Figure 7.14 depict the confusion matrix and metrics graphs.

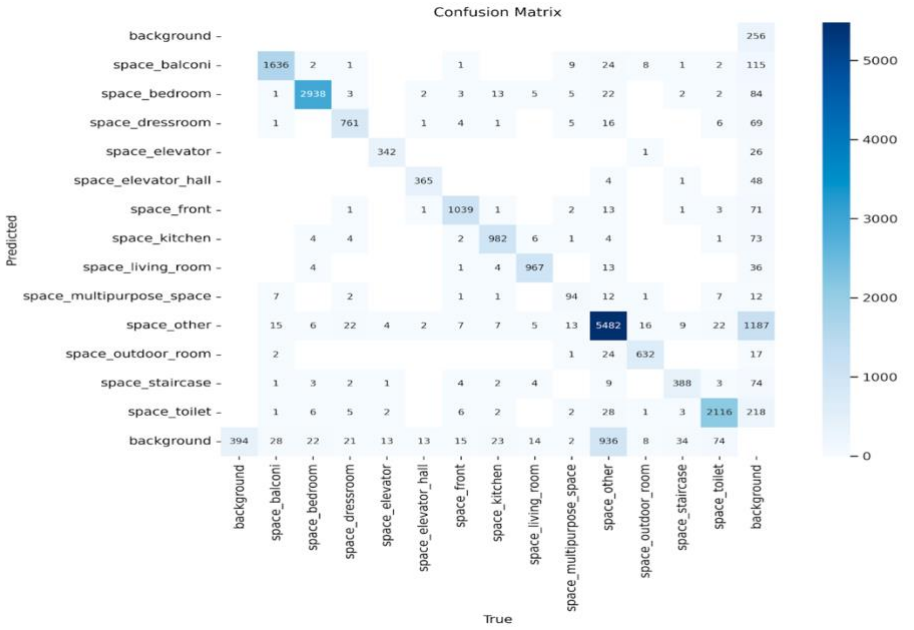


Fig. 7.13: Experiment 5 confusion matrix

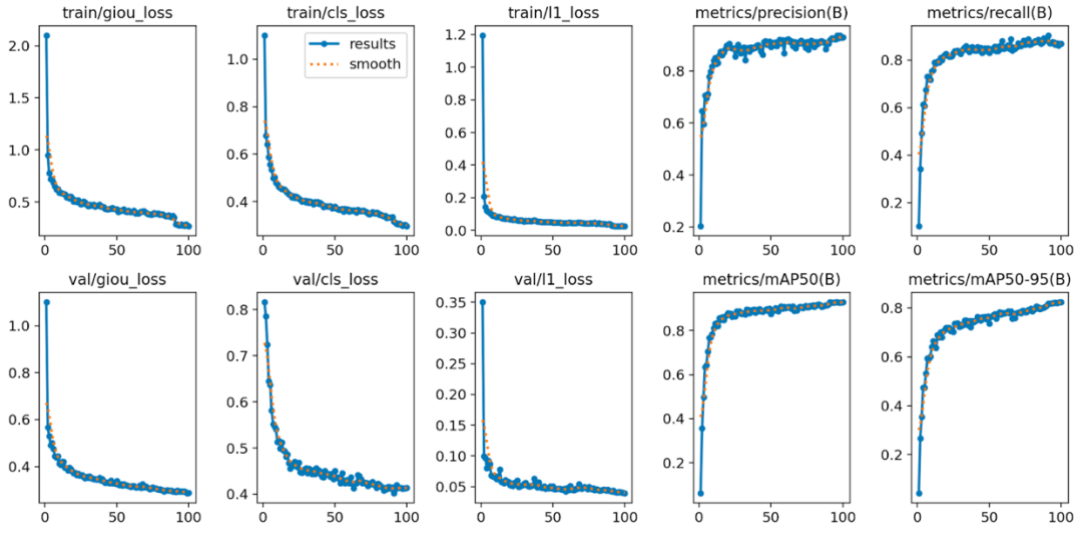


Fig. 7.14: Experiment 5 evaluation metrics graphs

Following Figure 7.15 illustrates the visualization of testing set results on Dataset-2.

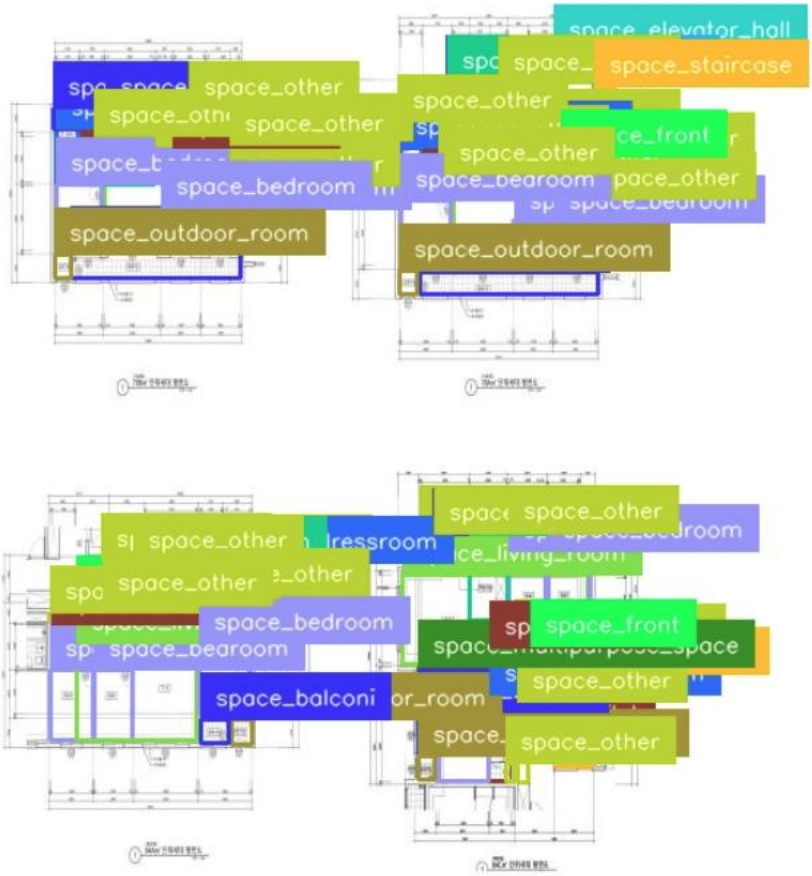


Fig. 7.15: Experiment 5 testing set visualization

7.4.6 Results From Experiment 6

Experiment 6, carried out on Dataset-3, utilized the RT-DETR Large object detection model and SAM VIT-H segmentation model, resulting in 96.2% of mAP, 94.7% of Precision, and 95.6% of Recall. The confusion matrix offers insights into the model's object detection capabilities, demonstrating enhanced accuracy rates across all 21 identified classes. Moreover, visualization graphs of evaluation metrics like box_loss, cls_loss, and dfl_loss were generated to illustrate model performance. These graphs exhibit smooth curves, indicating the satisfactory performance of the trained model.

Figure 7.16 depicts the experiment 6 confusion matrix. In Figure 7.17, we present the evaluation metrics graph for experiment 6.

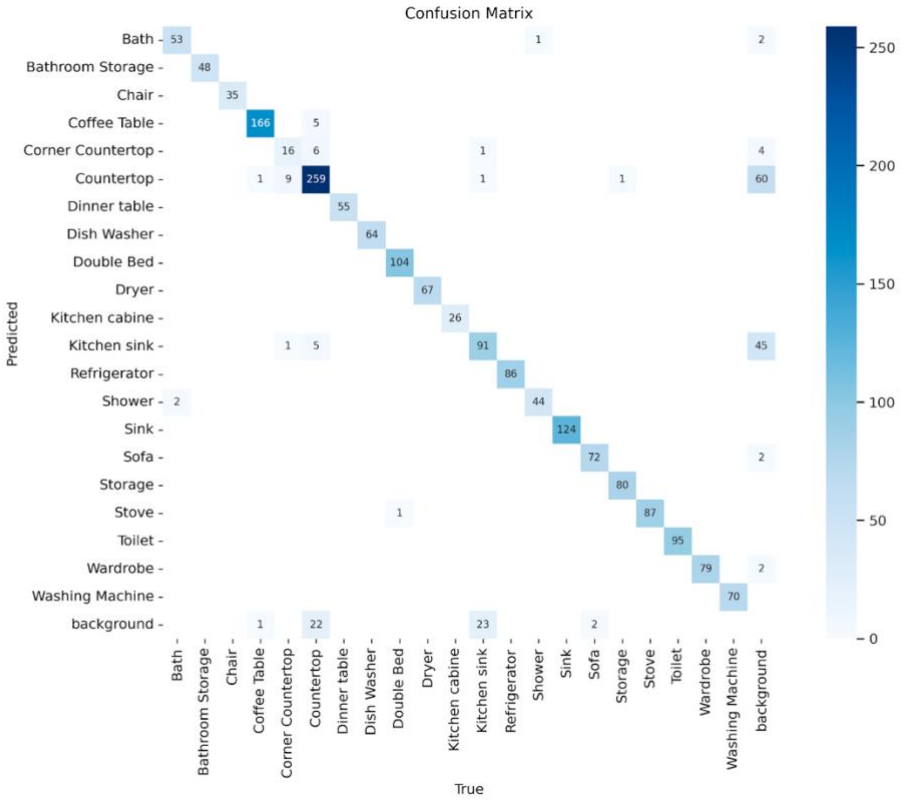


Fig. 7.16: Experiment 6 confusion matrix

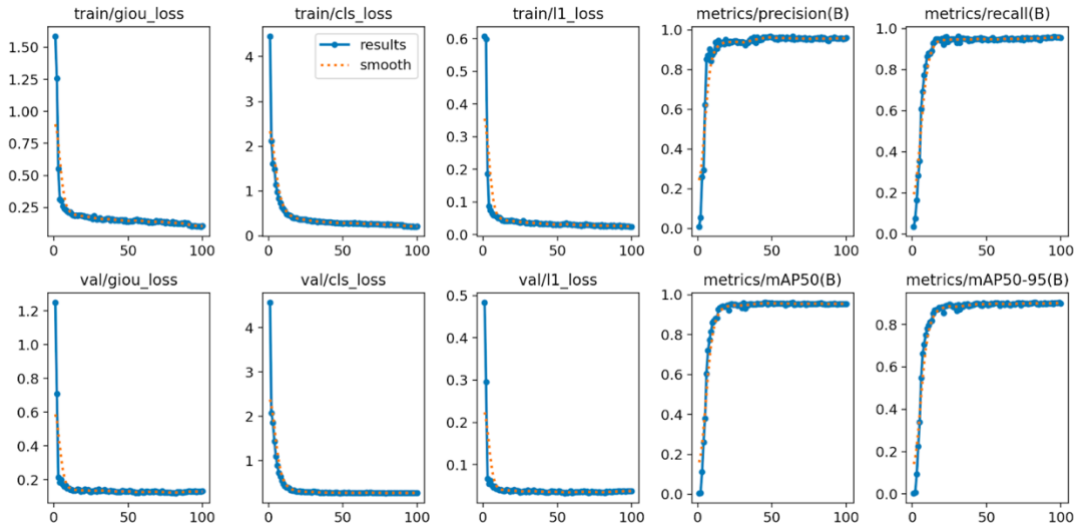


Fig. 7.17: Experiment 6 evaluation metrics graphs

Figure 7.18 illustrates the visualization of testing set results on Dataset-3.

Fig. 7.18: Experiment 6 testing set visualization

7.5 Discussion on Experiments

Based on the experiments on Dataset-2 and Dataset-3, which encompass a broader range of object classes, we observed that the RT-DETR model achieves significantly faster inference times per image than the YOLOv8 model. However, despite the speed advantage, observations from the six experiments that were conducted revealed distinct characteristics in terms of usability for subsequent processing steps. Both the YOLOv8 and RT-DETR models demonstrated comprehensive support for utilizing their bounding box predictions as visual prompts for the Segment Anything Model. This is attributed to the well-structured prediction output provided by both models. Although the RT-DETR provided a set of encoded numerical arrays, we could extract the correct detections from it accurately. Consequently, despite the faster inference times exhibited by the RT-DETR model, we had to perform inference locally without any deployment compared to YOLOv8, which could deploy on Roboflow.

Without clear visual prompts, the Segment Anything Model tends to hallucinate object identification in floor plan images. However, by utilizing object detection models, we guide the model to create masks only for identified objects. This is particularly advantageous, as evidenced by the near-zero error rates in the segmented masks observed in the results. This demonstrates the effectiveness of leveraging object detection models to enhance the performance and robustness of the segmentation process in analyzing floor plans.

Figures 7.19, 7.20, 7.21, 7.22, 7.23, and 7.24 present the visualizations in both the detection and segmentation stages and how visual prompts lead to precise segmentation outputs.

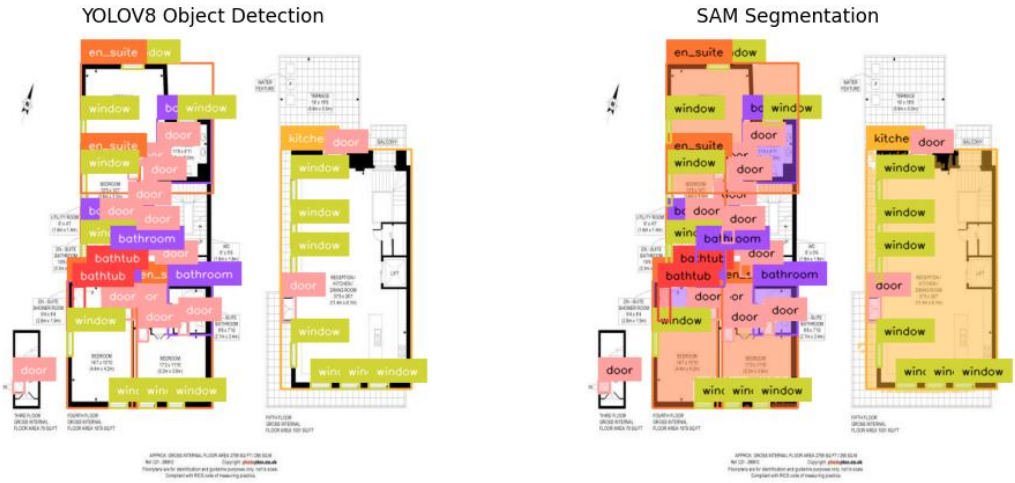


Fig. 7.19: Dataset-1 object detection and segmentation visualization

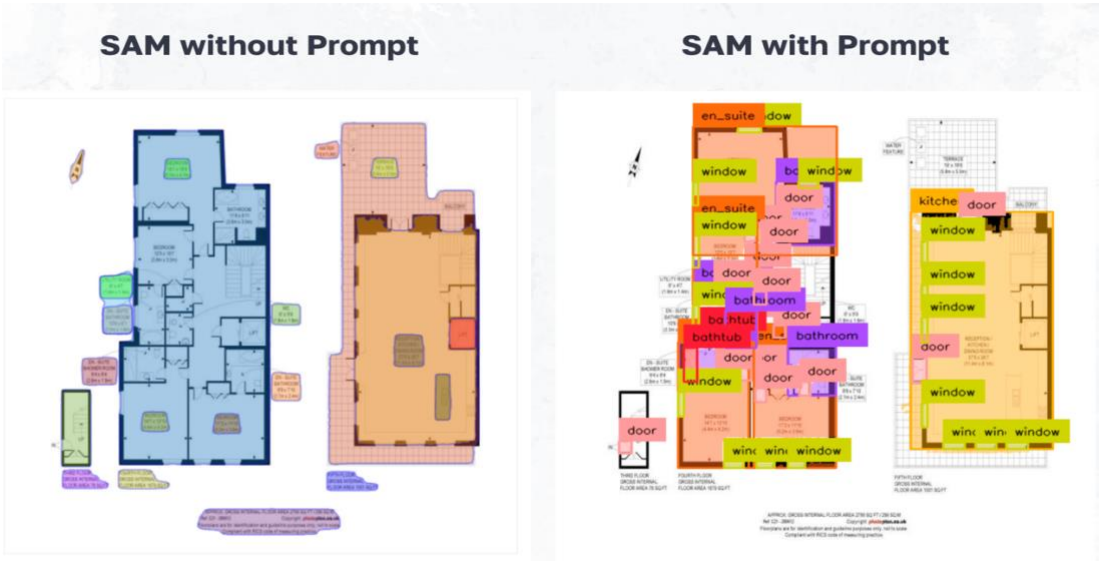


Fig. 7.20: Dataset-1 sam visualizations with prompts

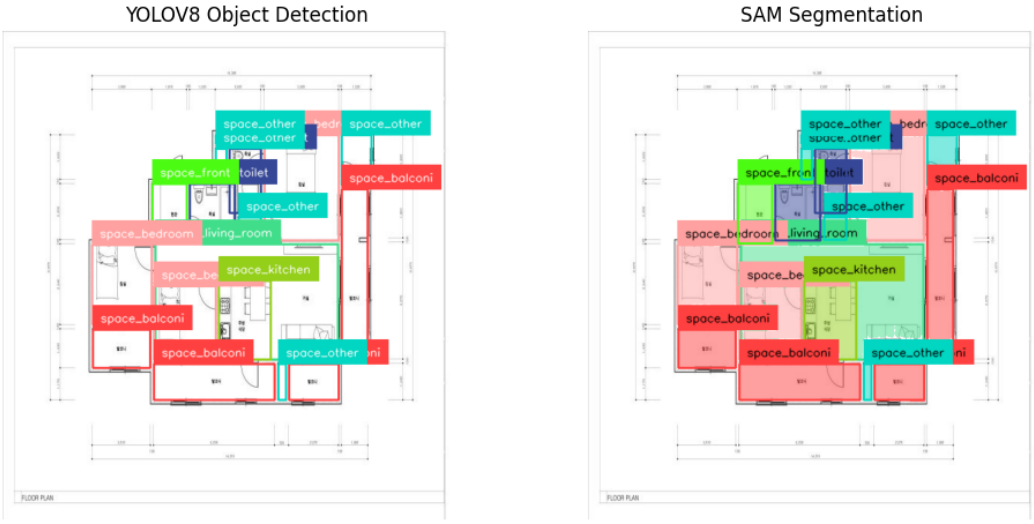
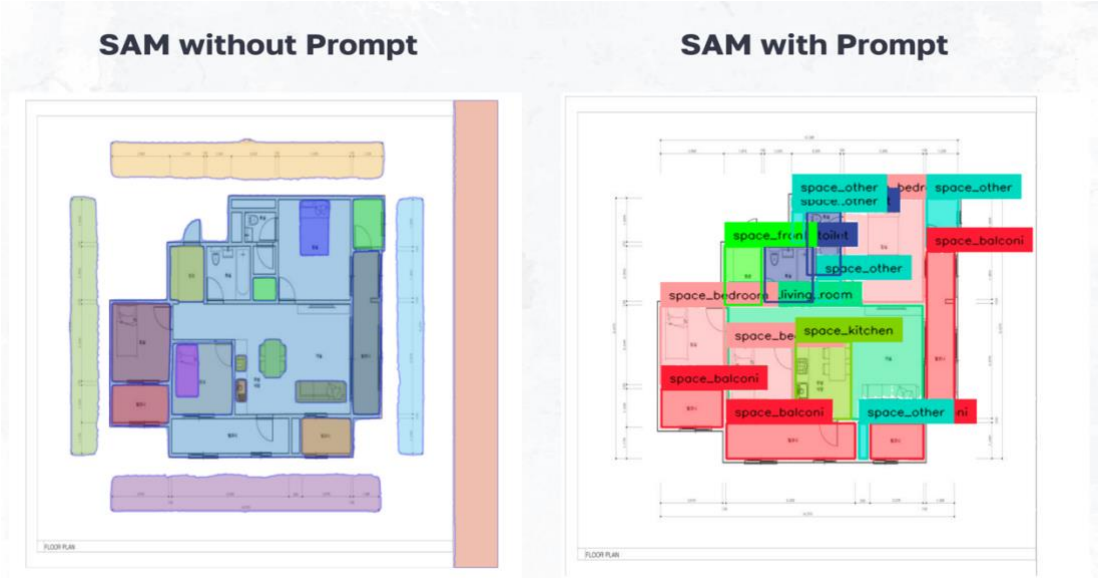


Fig. 7.21: Dataset-2 object detection and segmentation visualization



7.22: Dataset-2 sam visualizations with prompts

While the Segment Anything Model (SAM) demonstrates the ability to generate object masks based on bounding boxes provided by object detection models, it was not assessed using independent evaluation metrics. Since SAM relies on bounding boxes from a pre-trained object detection model, its segmentation accuracy is inherently tied to the object detection model's performance. Evaluating SAM with metrics like IoU and PQ would essentially re-evaluate the object detection stage, providing less insight into the overall system.

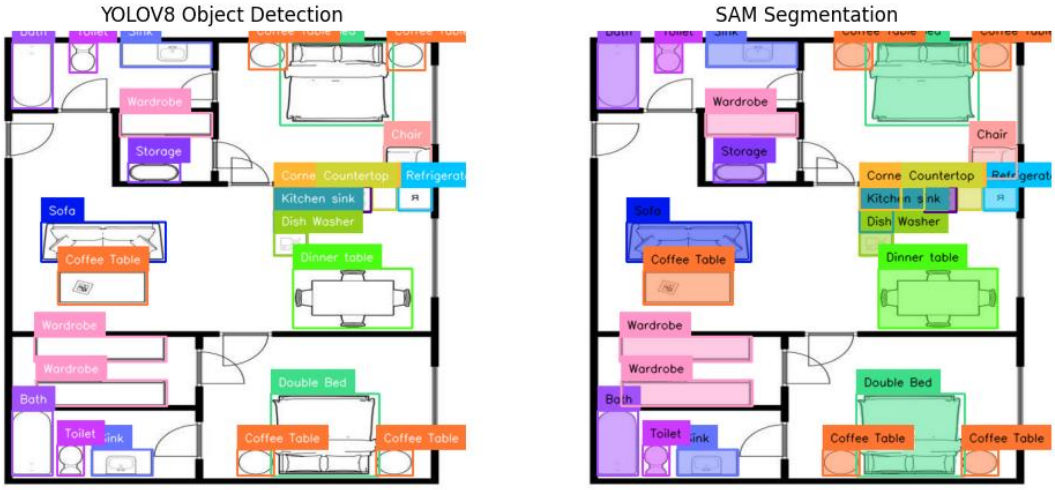


Fig. 7.23: Dataset-3 object detection and segmentation visualization

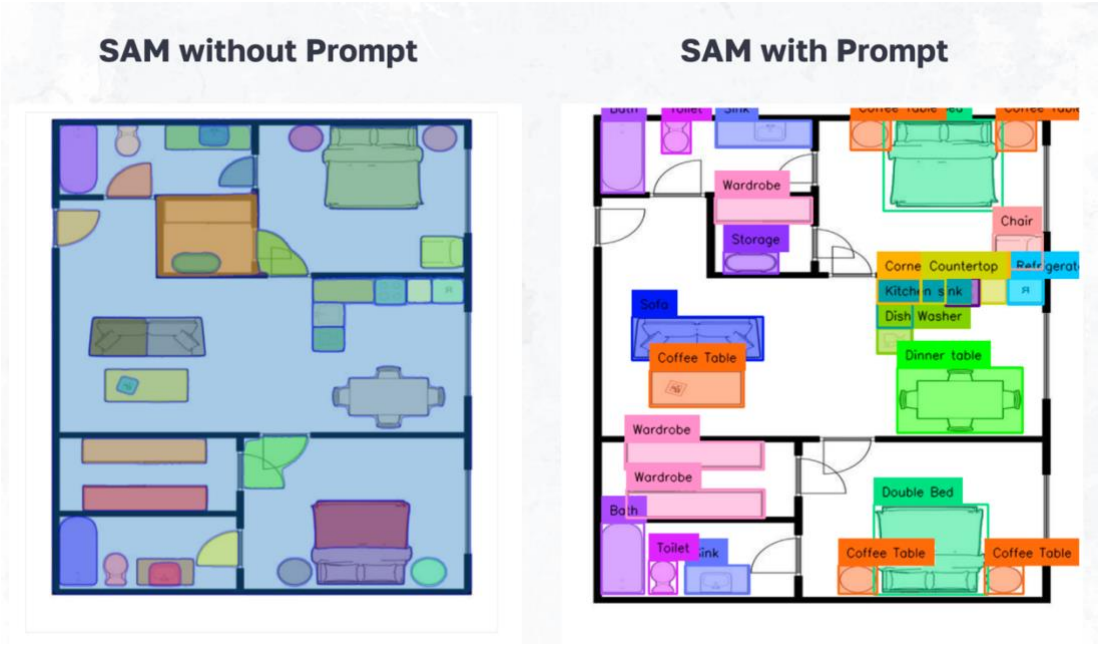


Fig. 7.24: Dataset-3 sam visualizations with prompts

7.6 Summary

In this evaluation chapter, we outlined the process using specific metrics. By analyzing mAP, precision, and recall, we gained insights into the system's capabilities and identified strengths and weaknesses. Moreover, through a comparative analysis conducted on three benchmark datasets, we assessed the effectiveness of our proposed approach.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Introduction

The prior chapter delves into the evaluation process conducted on the BuilderFormer. This chapter serves as a culmination of the entire research project, offering an overarching conclusion. It provides an overview of the project results to the targeted objectives, discusses any limitations encountered during the research process, and highlights potential opportunities for further research that have been identified.

8.2 Overall Conclusion

At the outset of the research project, the hypothesis was posited that the proposed panoptic segmentation approach using vision transformers for floor plan analysis would be an effective solution compared to traditional supervised and unsupervised methods. The BuilderFormer application was established using vision transformers as the primary technology to validate the hypothesis.

The research commenced with a comprehensive literature review, comparing existing approaches in automatically analyzing floor plans and exploring the applicability of vision transformers in mitigating limitations observed in prior studies. Considerable dedication was allocated to comprehending the principles underlying panoptic segmentation with vision transformers, drawing insights from authoritative reference materials. Subsequently, an approach centered around visual prompting, supplemented by the SAM, was meticulously formulated as the primary technique for generating segmentation masks corresponding to identified objects in floor plans. This process was underpinned by the design and development of modules and their interactions, progressing through iterative stages of implementation. Chapter 07 discussed the evaluation process and its findings, which involved assessing the proposed approach against several evaluation criteria established by the researcher. The evaluation results indicated that the development enhanced the target process, thus achieving the project's aim and objectives.

8.3 Objective Wise Achievements

Table 8.1 focuses on the project's current status and the relevant sections of the project outcomes compared to the project objectives.

TABLE 8.1: Objective Wise Achievements

Objectives	Status	Chapters Related
Critical review of methodologies used in automatic floor plan analysis.	Completed	Chapter 02
In-depth study of technologies used in automatic floor plan analysis.	Completed	Chapter 04, Chapter 05
Design and development of a panoptic segmentation approach to automatic floor plan analysis using vision transformers.	Completed	Chapter 03, Chapter 06
Evaluate the developed solution using proper evaluation metrics.	Completed	Chapter 07

All of the objectives mentioned above were accomplished within the research project.

8.4 Limitations

One significant limitation we faced pertained to the computational resources required for processing larger datasets within the floor plan domain. Despite encountering a dataset comprising 15,000 images, our research was constrained by the limited GPU environment. As a result, we could only experiment with datasets containing a maximum of 5000 images. This restricted access to high-end GPUs hindered our ability to fully explore and analyze larger datasets, potentially limiting the scope and depth of our research findings. Another limitation related to the RT-DETR model is its increased demand for memory when handling larger datasets. Consequently, for Dataset-2, we had to adjust the batch size slightly during training to avoid memory issues.

8.5 Future Work

In future endeavors, accessing more powerful computational resources will be essential for conducting comprehensive analysis on larger-scale datasets within the floor plan domain. Furthermore, we aim to extend our approach by experimenting with various floor plan styles, including those of different building types. Thus far, our focus has been primarily on indoor plans. In addition, we will be partnering with BuilderBid to continue our work by integrating this research into their production-grade software to streamline the floor plan takeoff process.

8.6 Summary

This chapter outlines the overall conclusion, objective achievements, limitations, and future work of the BuilderFormer. It identifies the constraints encountered during the development process and discusses potential solutions. Additionally, it delineates further enhancements required to augment the usefulness and efficacy of the BuilderFormer, defining them as areas for future work.

REFERENCES

[1] "Raster-to-Vector: Revisiting Floorplan Transformation | IEEE Conference Publication | IEEE Xplore." Accessed: Jun. 11, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/8237503

[2] L. Gimenez, S. Robert, F. Suard, and K. Zreik, "Automatic reconstruction of 3D building models from scanned 2D floor plans," *Autom. Constr.*, vol. 63, pp. 48–56, Mar. 2016, doi: 10.1016/j.autcon.2015.12.008.

[3] J. Yang, H. Jang, J. Kim, and J. Kim, "Semantic Segmentation in Architectural Floor Plans for Detecting Walls and Doors," in *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, Oct. 2018, pp. 1–9. doi: 10.1109/CISP-BMEI.2018.8633243.

[4] "How To Do A Construction Takeoff | ProEst." Accessed: Aug. 27, 2023. [Online]. Available: https://proest.com/construction/takeoffs/how-to-do-a-takeoff/

[5] S. Macé, H. Locteau, E. Valveny, and S. Tabbone, "A system to detect rooms in architectural floor plan images," in *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, in DAS '10. New York, NY, USA: Association for Computing Machinery, Jun. 2010, pp. 167–174. doi: 10.1145/1815330.1815352.

[6] S. Kim, S. Park, H. Kim, and K. Yu, "Deep Floor Plan Analysis for Complicated Drawings Based on Style Transfer," *J. Comput. Civ. Eng.*, vol. 35, no. 2, p. 04020066, Mar. 2021, doi: 10.1061/(ASCE)CP.1943-5487.0000942.

[7] A. Vaswani *et al.*, "Attention Is All You Need." arXiv, Dec. 05, 2017. doi: 10.48550/arXiv.1706.03762.

[8] P. N. Pizarro, N. Hitschfeld, I. Sipiran, and J. M. Saavedra, "Automatic floor plan analysis and recognition," *Autom. Constr.*, vol. 140, p. 104348, Aug. 2022, doi: 10.1016/j.autcon.2022.104348.

[9] W. Lv *et al.*, "DETRs Beat YOLOs on Real-time Object Detection." arXiv, Jul. 06, 2023. Accessed: Jan. 15, 2024. [Online]. Available: http://arxiv.org/abs/2304.08069

[10] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollar, "Panoptic Segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, Jun. 2019, pp. 9396–9405. doi: 10.1109/CVPR.2019.00963.

[11] B. Cheng, A. G. Schwing, and A. Kirillov, "Per-Pixel Classification is Not All You Need for Semantic Segmentation." arXiv, Oct. 31, 2021. Accessed: Jun. 01, 2023. [Online]. Available: http://arxiv.org/abs/2107.06278

[12] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, "Masked-attention Mask Transformer for Universal Image Segmentation," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, New Orleans, LA, USA: IEEE, Jun. 2022, pp. 1280–1289. doi: 10.1109/CVPR52688.2022.00135.

[13] Z. Fan, T. Chen, P. Wang, and Z. Wang, "CADTransformer: Panoptic Symbol Spotting Transformer for CAD Drawings," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, New Orleans, LA, USA: IEEE, Jun. 2022, pp. 10976–10986. doi: 10.1109/CVPR52688.2022.01071.

[14] "Takeoff & Estimating Software for Builders | BuilderBid." Accessed: Aug. 27, 2023. [Online]. Available: https://builderbid.com/

[15] J. Cherneff, R. Logcher, J. Connor, and N. Patrikalakis, "Knowledge-based interpretation of architectural drawings," *Res. Eng. Des.*, vol. 3, no. 4, pp. 195–210, Dec. 1992, doi: 10.1007/BF01580842.

[16] K. Ryall, S. Shieber, J. Marks, and M. Mazer, "Semi-automatic delineation of regions in floor plans," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Aug. 1995, pp. 964–969 vol.2. doi: 10.1109/ICDAR.1995.602062.

[17] L.-P. de las Heras, D. Fernández, E. Valveny, J. Lladós, and G. Sánchez, "Unsupervised Wall Detector in Architectural Floor Plans," in *2013 12th International Conference on Document Analysis and Recognition*, Aug. 2013, pp. 1245–1249. doi: 10.1109/ICDAR.2013.252.

[18] A. Barducci and S. Marinai, "Object recognition in floor plans by graphs of white connected components," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, Nov. 2012, pp. 298–301.

[19] L.-P. de las Heras and G. Sánchez, "And-Or Graph Grammar for Architectural Floor Plan Representation, Learning and Recognition. A Semantic, Structural and Hierarchical Model," in *Pattern Recognition and Image Analysis*, J. Vitrià, J. M. Sanches, and M. Hernández, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 17–24. doi: 10.1007/978-3-642-21257-4_3.

[20] L.-P. de las Heras, S. Ahmed, M. Liwicki, E. Valveny, and G. Sánchez, "Statistical segmentation and structural recognition for floor plan interpretation," *Int. J. Doc. Anal. Recognit. IJDAR*, vol. 17, no. 3, pp. 221–237, Sep. 2014, doi: 10.1007/s10032-013-0215-2.

[21] "Alpha shape based design space decomposition for island failure regions in reliability based design | Structural and Multidisciplinary Optimization." Accessed: Apr. 12, 2024. [Online]. Available: https://link.springer.com/article/10.1007/s00158-014-1224-6

[22] X. Guo and Y. Peng, "Floor Plan Classification Based on Transfer Learning," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, Dec. 2018, pp. 1720–1724. doi: 10.1109/CompComm.2018.8780679.

[23] X. Lv, S. Zhao, X. Yu, and B. Zhao, "Residential floor plan recognition and reconstruction," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA: IEEE, Jun. 2021, pp. 16712–16721. doi: 10.1109/CVPR46437.2021.01644.

[24] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y.

Weiss, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 833–851. doi: 10.1007/978-3-030-01234-2_49.

[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 580–587. doi: 10.1109/CVPR.2014.81.

[26] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 8798–8807. doi: 10.1109/CVPR.2018.00917.

[27] Z. Zeng, X. Li, Y. K. Yu, and C.-W. Fu, "Deep Floor Plan Recognition Using a Multi-Task Network With Room-Boundary-Guided Attention," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South): IEEE, Oct. 2019, pp. 9095–9103. doi: 10.1109/ICCV.2019.00919.

[28] C. P. Simonsen, F. M. Thiesson, M. P. Philipsen, and T. B. Moeslund, "Generalizing Floor Plans Using Graph Neural Networks," in *2021 IEEE International Conference on Image Processing (ICIP)*, Sep. 2021, pp. 654–658. doi: 10.1109/ICIP42928.2021.9506514.

[29] Y. Zhang, Y. He, S. Zhu, and X. Di, "The Direction-Aware, Learnable, Additive Kernels and the Adversarial Network for Deep Floor Plan Recognition." arXiv, Jan. 30, 2020. doi: 10.48550/arXiv.2001.11194.

[30] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets." arXiv, Nov. 06, 2014. Accessed: Jun. 20, 2023. [Online]. Available: http://arxiv.org/abs/1411.1784

[31] W. Huang and H. Zheng, *Architectural Drawings Recognition and Generation through Machine Learning*. 2018. doi: 10.52842/conf.acadia.2018.156.

[32] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." arXiv, Jun. 03, 2021. doi: 10.48550/arXiv.2010.11929.

[33] A. Kirillov *et al.*, "Segment Anything." arXiv, Apr. 05, 2023. Accessed: Feb. 17, 2024. [Online]. Available: http://arxiv.org/abs/2304.02643

[34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection." arXiv, May 09, 2016. doi: 10.48550/arXiv.1506.02640.

[35] Ultralytics, "YOLOv8." Accessed: Apr. 26, 2024. [Online]. Available: https://docs.ultralytics.com/models/yolov8

[36] "Roboflow: Computer vision tools for developers and enterprises." Accessed: Apr. 25, 2024. [Online]. Available: https://roboflow.com/

[37] "Streamlit • A faster way to build and share data apps." Accessed: Apr. 26, 2024. [Online]. Available: https://streamlit.io/

[38] "Albumentations: Efficient Image Augmentation Library for Machine Learning." Accessed: Apr. 26, 2024. [Online]. Available: https://albumentations.ai

APPENDIX - A

CODE FOR WEB APP

```python
# Import required libraries
from inference import get_model
import numpy as np
import PIL
from segment_anything import sam_model_registry, SamPredictor
import streamlit as st
import supervision as sv
import torch

# Load SAM model
CHECKPOINT_PATH = "sam_vit_h_4b8939.pth";
DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
MODEL_TYPE = "vit_h"
sam = sam_model_registry[MODEL_TYPE](checkpoint=CHECKPOINT_PATH)
sam.to(device=DEVICE)


# SAM function to create masks
1 usage
def segment(sam_predictor: SamPredictor, image: np.ndarray, xyxy: np.ndarray) -> np.ndarray:
    sam_predictor.set_image(image)
    result_masks = []
    for box in xyxy:
        masks, scores, logits = sam_predictor.predict(box=box, multimask_output=True)
        index = np.argmax(scores)
        result_masks.append(masks[index])
    return np.array(result_masks)


# Dropdown options for different takeoffs
dropdown_options = ["Properties Only", "Room Spaces Only", "Furniture Types Only"]

# Setting page layout
st.set_page_config(
    page_title="BUILDERFORMER",
    layout="wide",
    initial_sidebar_state="expanded",
)

```

```python
# Creating sidebar
with st.sidebar:
    # Adding file uploader to sidebar for selecting images
    source_img = st.file_uploader(
        "Upload an image...", type=("jpg", "jpeg", "png", 'bmp', 'webp'))

    if source_img:
        # Opening the uploaded image
        uploaded_image = PIL.Image.open(source_img)

    # Create the dropdown
    selected_dropdown_option = st.selectbox("Select what you want to detect...", dropdown_options)

# Creating main page heading
st.title("BUILDERFORMER")
# Creating two columns on the main page
col1, col2 = st.columns(2)

if st.sidebar.button('Detect Objects'):
    # Get selected class labels based on dropdown selection
    if selected_dropdown_option == dropdown_options[0]:
        try:
            model = get_model(model_id="builderformer/9")
            # Perform object detection with YOLO
            results = model.infer(uploaded_image)
            detections = sv.Detections.from_inference(results[0].dict(by_alias=True, exclude_none=True))
        except Exception as ex:
            st.error(
                f"Unable to load model.")
            st.error(ex)
    elif selected_dropdown_option == dropdown_options[1]:
        try:
            model = get_model(model_id="builderformer-4/2")
            # Perform object detection with YOLO
            results = model.infer(uploaded_image)
            detections = sv.Detections.from_inference(results[0].dict(by_alias=True, exclude_none=True))
        except Exception as ex:
            st.error(
                f"Unable to load model.")
            st.error(ex)
```

```
    # Create the dropdown
    selected_dropdown_option = st.selectbox("Select what you want to detect...", dropdown_options)

# Creating main page heading
st.title("BUILDERFORMER")
# Creating two columns on the main page
col1, col2 = st.columns(2)

if st.sidebar.button('Detect Objects'):
    # Get selected class labels based on dropdown selection
    if selected_dropdown_option == dropdown_options[0]:
        try:
            model = get_model(model_id="builderformer/9")
            # Perform object detection with YOLO
            results = model.infer(uploaded_image)
            detections = sv.Detections.from_inference(results[0].dict(by_alias=True, exclude_none=True))
        except Exception as ex:
            st.error(
                f"Unable to load model.")
            st.error(ex)
    elif selected_dropdown_option == dropdown_options[1]:
        try:
            model = get_model(model_id="builderformer-4/2")
            # Perform object detection with YOLO
            results = model.infer(uploaded_image)
            detections = sv.Detections.from_inference(results[0].dict(by_alias=True, exclude_none=True))
        except Exception as ex:
            st.error(
                f"Unable to load model.")
            st.error(ex)
    else:
        try:
            model = get_model(model_id="builderformer-3/1")
            # Perform object detection with YOLO
            results = model.infer(uploaded_image)
            detections = sv.Detections.from_inference(results[0].dict(by_alias=True, exclude_none=True))
        except Exception as ex:
            st.error(
                f"Unable to load model.")
            st.error(ex)
```

```
                f"Unable to load model.")
            st.error(ex)

    # Plot the detected objects
    if detections:
        # Create supervision annotators
        bounding_box_annotator = sv.BoundingBoxAnnotator()
        label_annotator = sv.LabelAnnotator()

        # Annotate the image with inference results
        annotated_image = bounding_box_annotator.annotate(
            scene=uploaded_image, detections=detections)
        annotated_image = label_annotator.annotate(
            scene=annotated_image, detections=detections)
        with col1:
            st.image(annotated_image,
                     caption='Detected Objects',
                     use_column_width=True
                     )
    else:
        with col1:
            st.info("No selected objects detected in the image.")

    # Perform segmentation using SAM model
    if detections:
        mask_predictor = SamPredictor(sam)
        image_array = np.array(uploaded_image)
        detections.mask = segment(sam_predictor=mask_predictor,image=image_array,xyxy=detections.xyxy)
        # Create supervision annotators
        mask_annotator = sv.MaskAnnotator()

        # Annotate the image with inference results
        annotated_image = mask_annotator.annotate(scene=uploaded_image, detections=detections)

        # Display the segmented image
        with col2:
            st.image(annotated_image, caption='Segmented Image', use_column_width=True)
    else:
        with col2:
            st.info("No segmentation performed since no objects were detected for selected category.")
```